

# Pixel Labeling

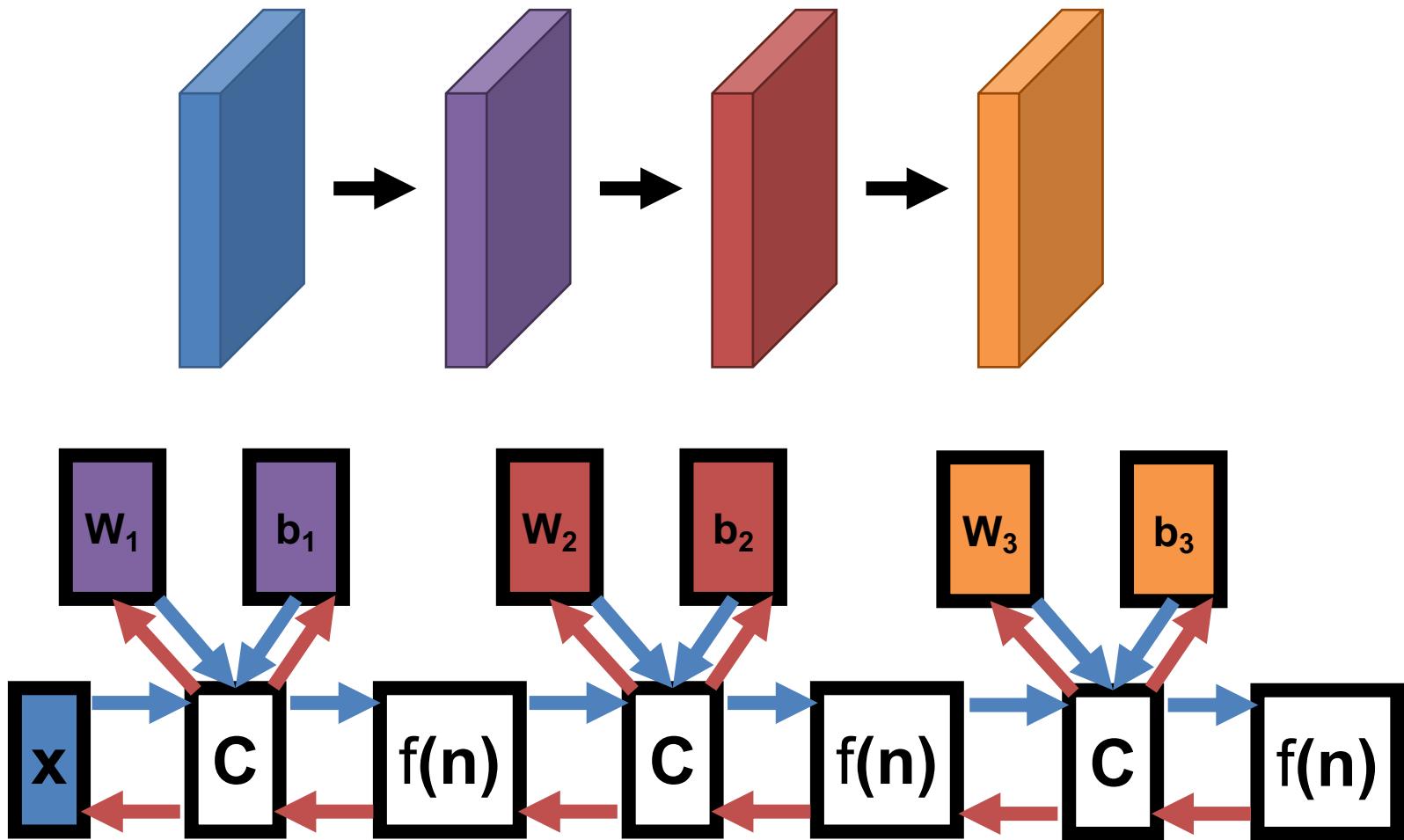
EECS 442 – Jeong Joon Park  
Winter 2024, University of Michigan

# Administrivia

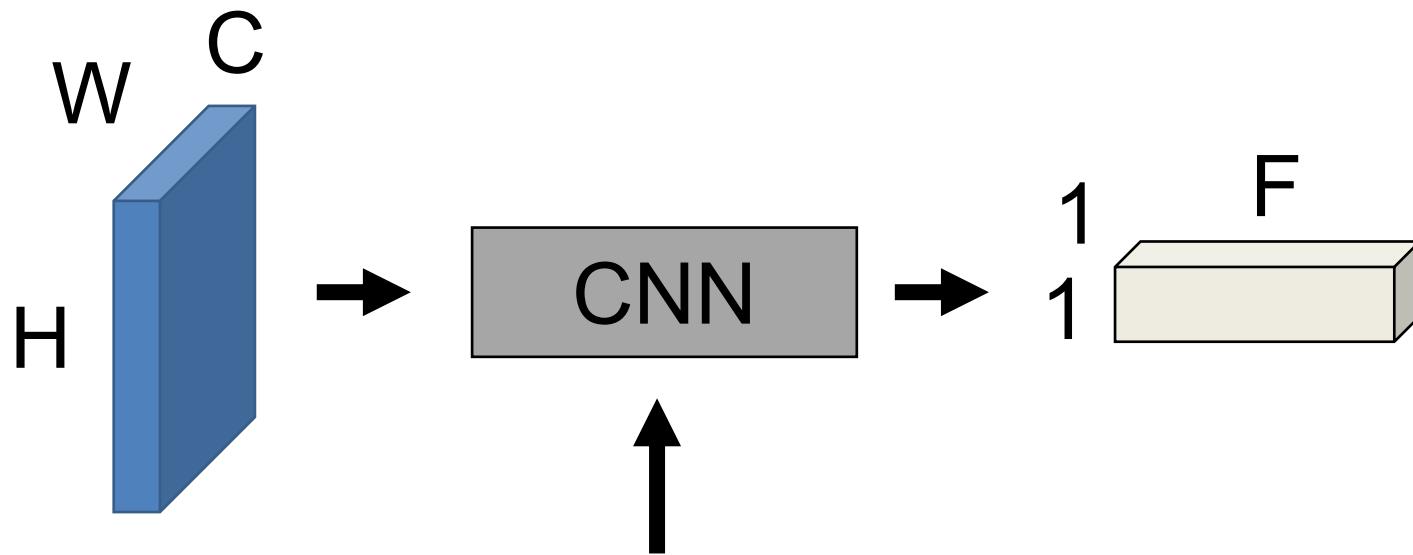
- Project team declaration: [link](#)  
Due this Wednesday. Email us if you have problem finding a team
- Briefly check your ideas at discussion/OH with GSIs or Me
- Project proposal (due Mar. 27<sup>th</sup>) – purely meant to help your scheduling
- Mid-term practice exam will be out on Wed

# Recap

# Convolutional Neural Network (CNN)



# Mental Model



Function of the image that is parameterized by the convolutional filter weights and biases. We design the form of the function and fit the parameters to data.

# Training a CNN

- Download a big dataset
- Initialize network weights randomly
- for epoch in range(epochs):
  - Shuffle dataset
  - for each minibatch in datsaet.:
    - Put data on GPU
    - Compute gradient with respect to loss
    - Update gradient with SGD

# Training a CNN from Scratch

Need to start  $w$  somewhere

- AlexNet: weights  $\sim \text{Normal}(0, 0.01)$ , bias = 1
- “Xavier” or “Kaiming” initialization: Initialize weights as a function of number of input/output channels

Take-home: important, but use defaults

# Training a ConvNet

- Convnets typically have millions of parameters:
  - AlexNet: 62 million
  - VGG16: 138 million
  - ConvNeXt-L: 198M
- Convnets typically fit on ~1.2 million images
- Remember least squares: if we have fewer data points than parameters, we're in trouble
- Solution: need regularization / more data

# Training a CNN – Weight Decay

SGD  
Update

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \epsilon \frac{\partial L}{\partial \mathbf{w}_t}$$

+Weight  
Decay

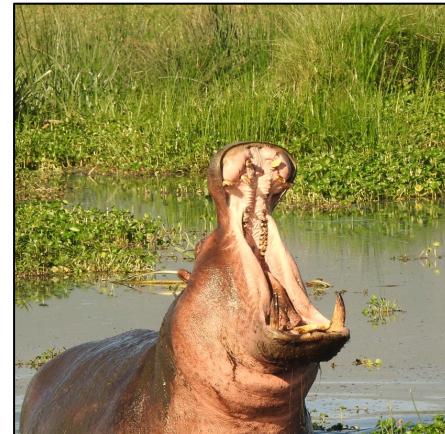
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \epsilon \mathbf{w}_t - \epsilon \frac{\partial L}{\partial \mathbf{w}_t}$$

## What does this remind you of?

Weight decay is similar to regularization but is not be the same for more complex optimization techniques.

See “Decoupled Weight Decay Regularization”, Loshchilov and Hutter.

# Augmentataion



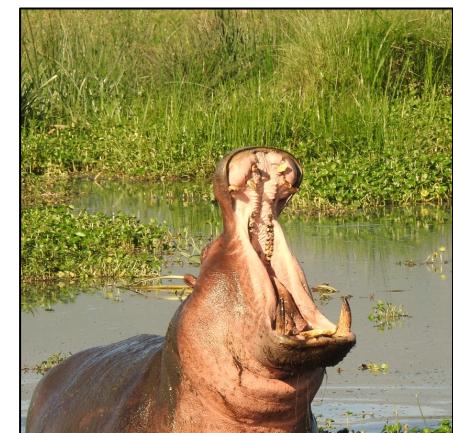
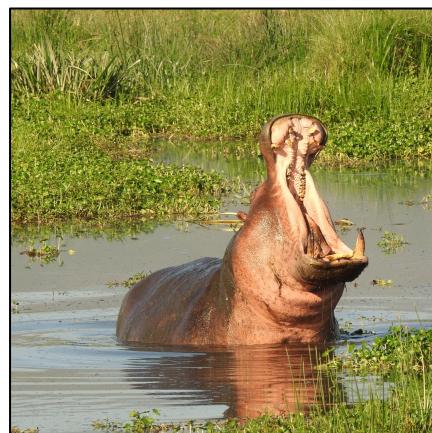
Horizontal  
Flip

Color  
Jitter

Image  
Cropping

# Training a CNN –Augmentation

- Apply transformations that don't affect the output
- Produces more data but you have to be careful that it doesn't change the meaning of the output

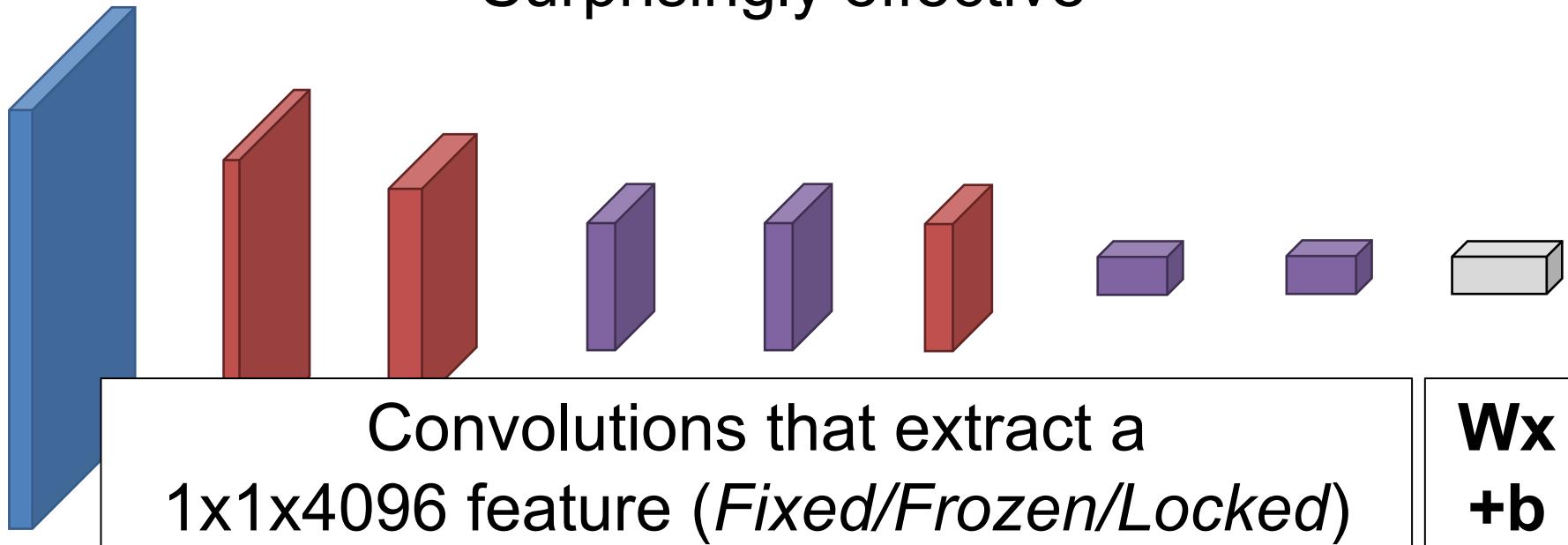


# Training a CNN – Fine-tuning

- What if you don't have data?
- Reuse weights learned from other data!
- (Often) Works like magic

# Fine-Tuning: Pre-trained Features

1. Extract some layer from an existing network
    2. Use as your new feature.
    3. Learn a linear model.
- Surprisingly effective



# Fine-Tuning: Transfer Learning

- Rather than initialize from random weights, initialize from some “pre-trained” model that does something else.
- Most common model is trained on ImageNet.
- Other pretraining tasks exist but are less popular.

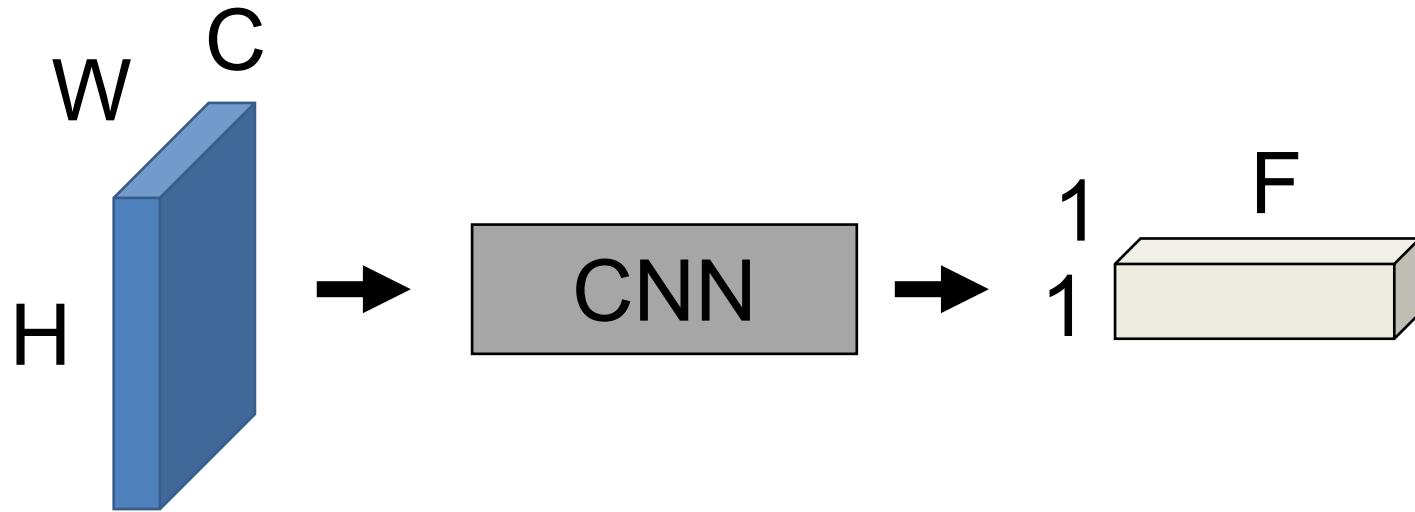
# Fine-Tuning: Transfer Learning

Why should this work?

Transferring from objects (dog) to scenes (waterfall)



# So Far



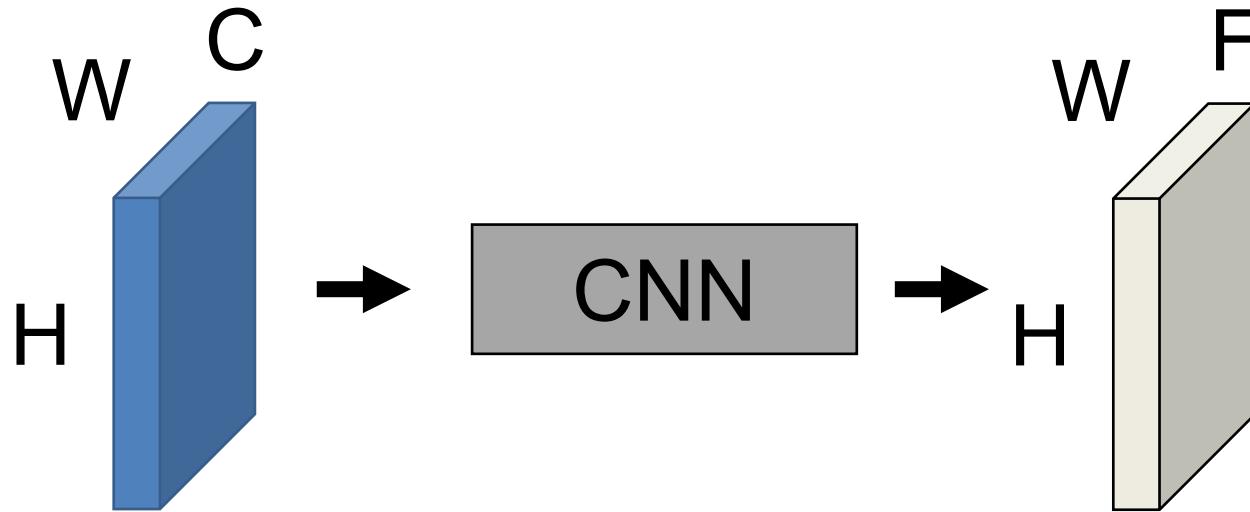
Convert  $H \times W$  image into a  $F$ -dimensional vector

Is this image a cat?

At what distance was this photo taken?

Is this image fake?

Now

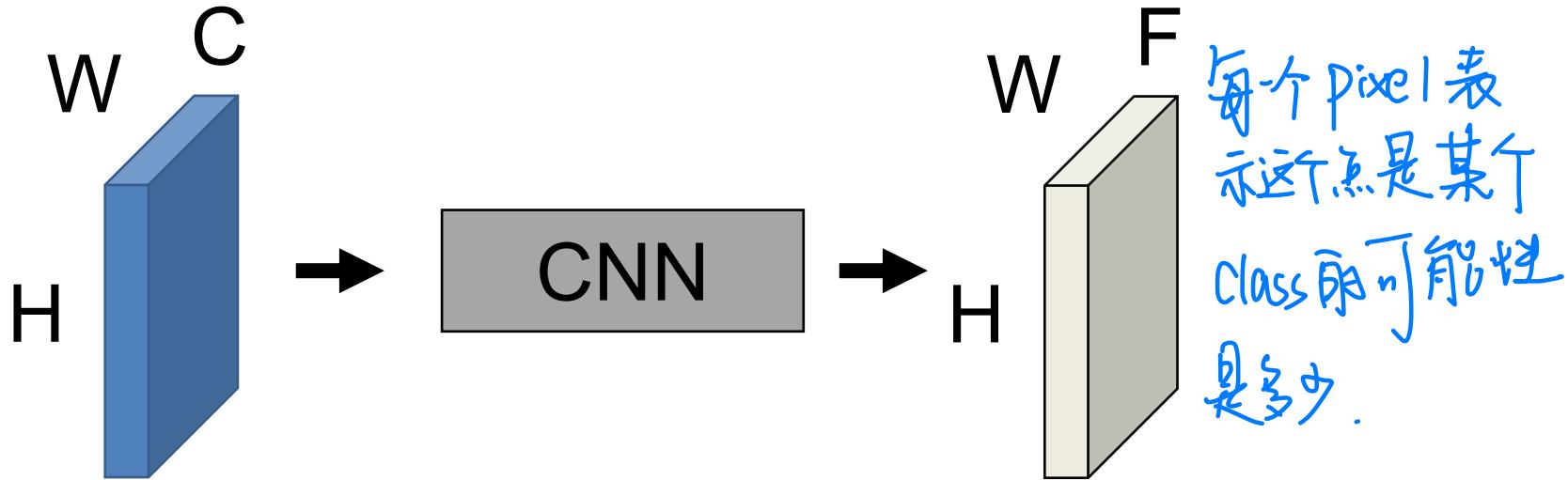


Convert  $H \times W$  image into a  $F$ -dimensional vector

- Which pixels in this image are a cat?
- How far is each pixel away from the camera?
- Which pixels of this image are fake?

现在不是一张图代表一个 Class j, 而是一张图中的某一部份代表一个 Class. 一张图可代表多个 Class.

# Semantic Segmentation



## Today's Running Example

- Predict  $F$ -dimensional vector representing probability of each of  $F$  classes at every pixel
- Loss computed/backprop'd at *every* pixel.

# Semantic Segmentation

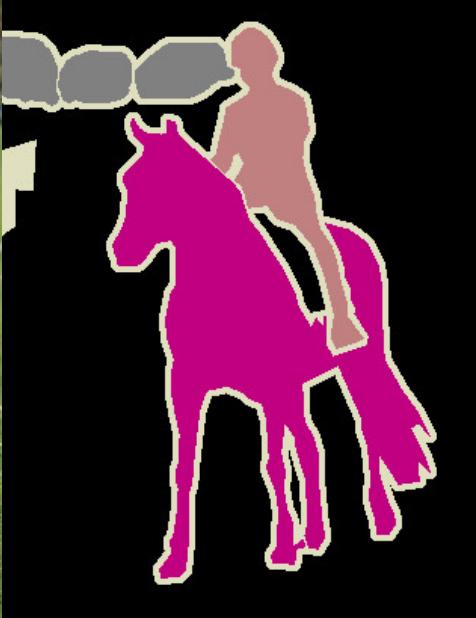
Each pixel has label, inc. **background**, and **unknown**  
Usually visualized by colors.

Note: don't distinguish between object *instances*

Input



Label



Input



Label



# Semantic Segmentation

“Semantic”: a usually meaningless word.  
Meant to indicate here that we’re **naming** things  
using language (instead of numbers, e.g.)

Input



Label



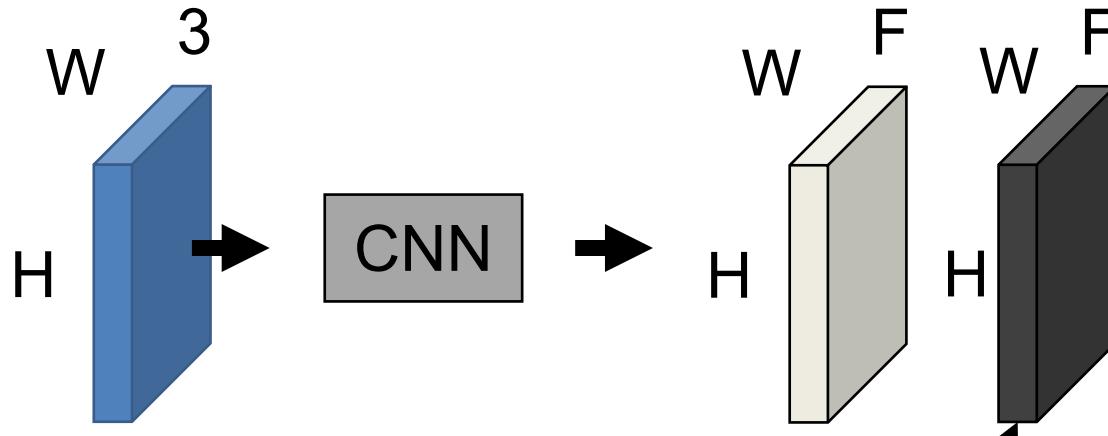
Input



Label



# Semantic Segmentation

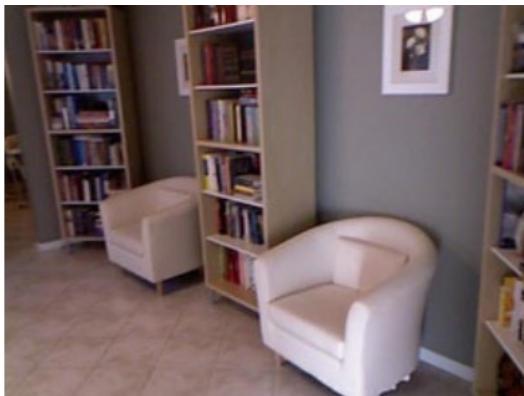


F-way classification  
loss function –  $\log\left(\frac{\exp((Wx)_{y_i})}{\sum_k \exp((Wx)_k)}\right)$   
at every pixel:

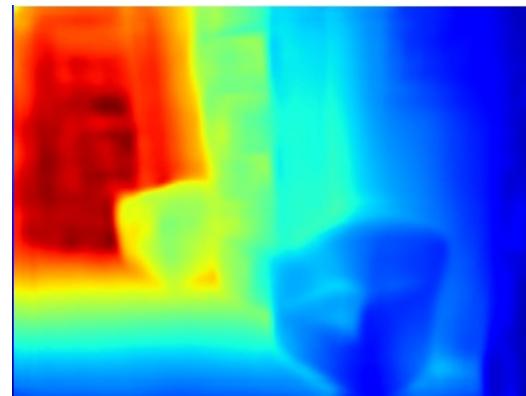
# Other Tasks – Depth Prediction

Instead: give label of depthmap, train network to do regression (e.g.,  $\|z_i - \hat{z}_i\|$  where  $z_i$  is the ground-truth and  $\hat{z}_i$  the prediction of the network at pixel i).

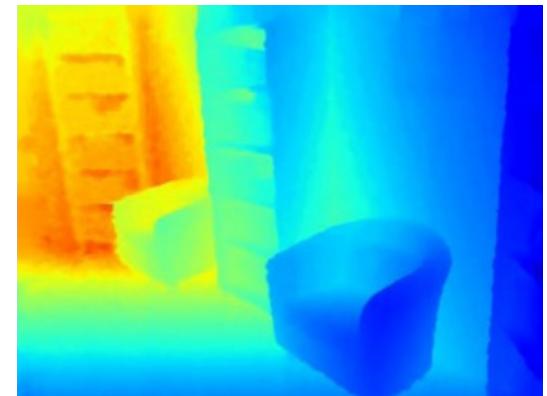
Input HxWx3  
RGB Image



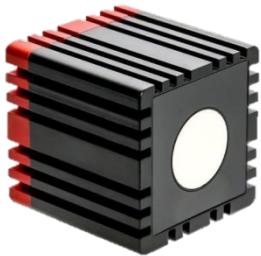
Output HxWx1  
Depth Image



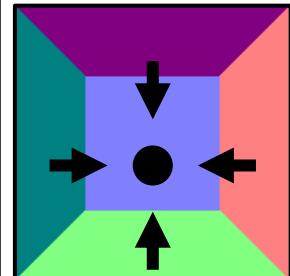
True HxWx1  
Depth Image



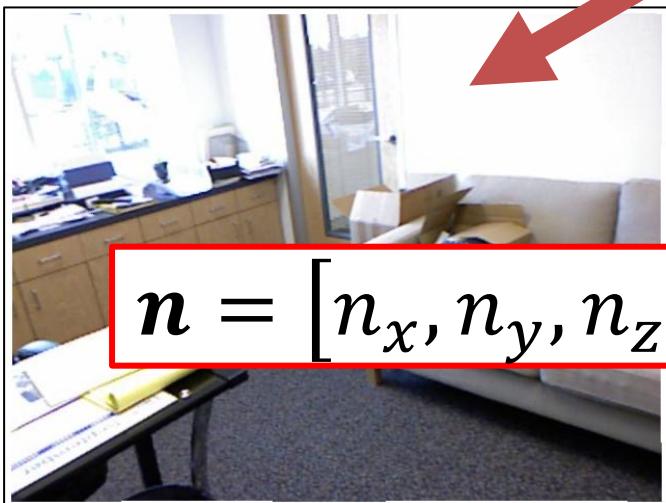
# Other Tasks – Surface Normals



Room

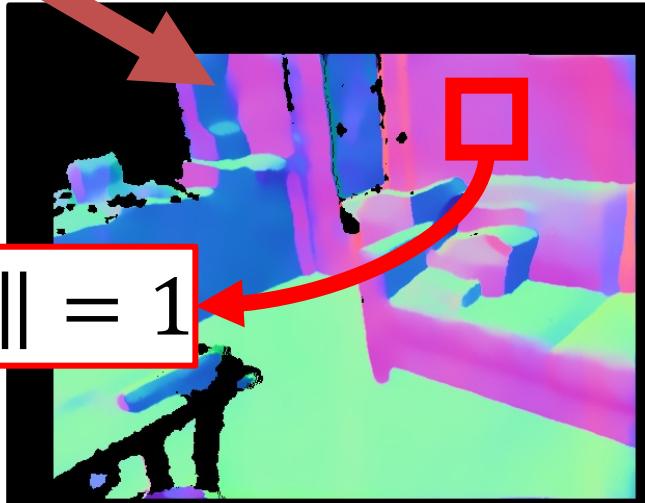


Legend



$$\mathbf{n} = [n_x, n_y, n_z], \|\mathbf{n}\| = 1$$

Color Image



Normals

# Surface Normals

Instead: train normal network to minimize  $\|\mathbf{n}_i - \widehat{\mathbf{n}}_i\|$   
where  $\mathbf{n}_i$  is ground-truth and  $\widehat{\mathbf{n}}_i$  prediction at pixel i.

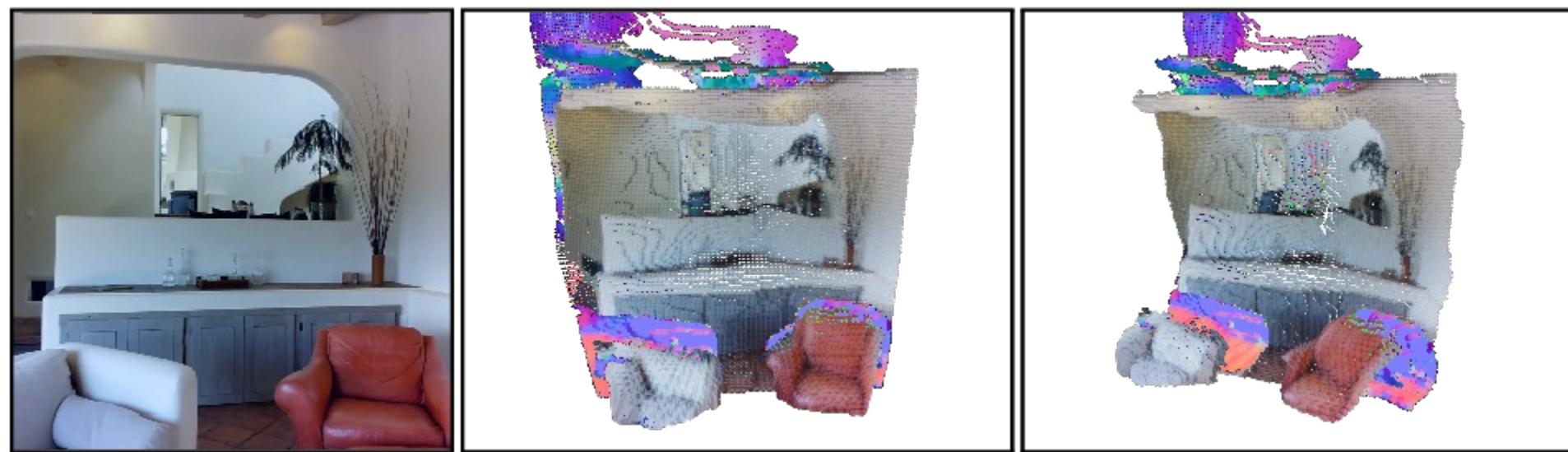
Input: HxWx3  
RGB Image



Output: HxWx3  
Normals



# 3D Reconstruction



Result credit: N. Kulkarni, J. Johnson, D.F. Fouhey, *What's Behind The Couch: Directed Ray Distance Functions for 3D Reconstruction.* ???, 2022.

# Other Tasks – Human Pose Estimation



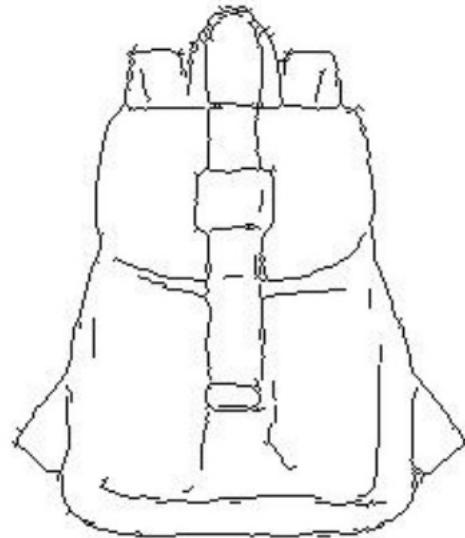
Source: <https://www.youtube.com/watch?v=2DiQUX11YaY>

Result credit: Z. Cao et al. *Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*. CVPR 2017.

# Other Task – Edges to Cats

Train network to minimize  $\|I_j - \hat{I}_j\|$  where  $I_j$  is GT and  $\hat{I}_j$  prediction at pixel j (*plus other magic*).

Input: HxWx1  
Sketch Image



Output: HxWx3  
Image



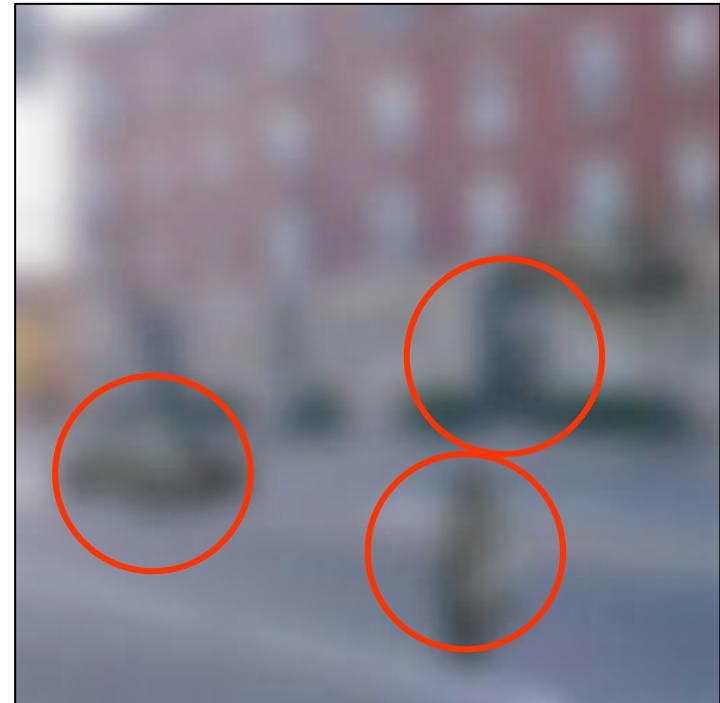
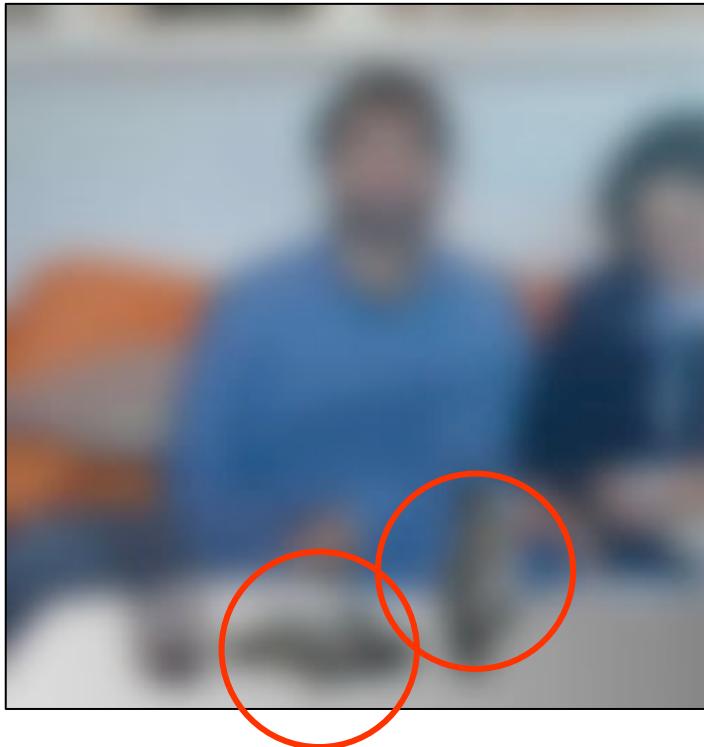
<https://affinelayer.com/pixsrv/>

Result credit: Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. CVPR 2017.

# Why Is This Task Hard?



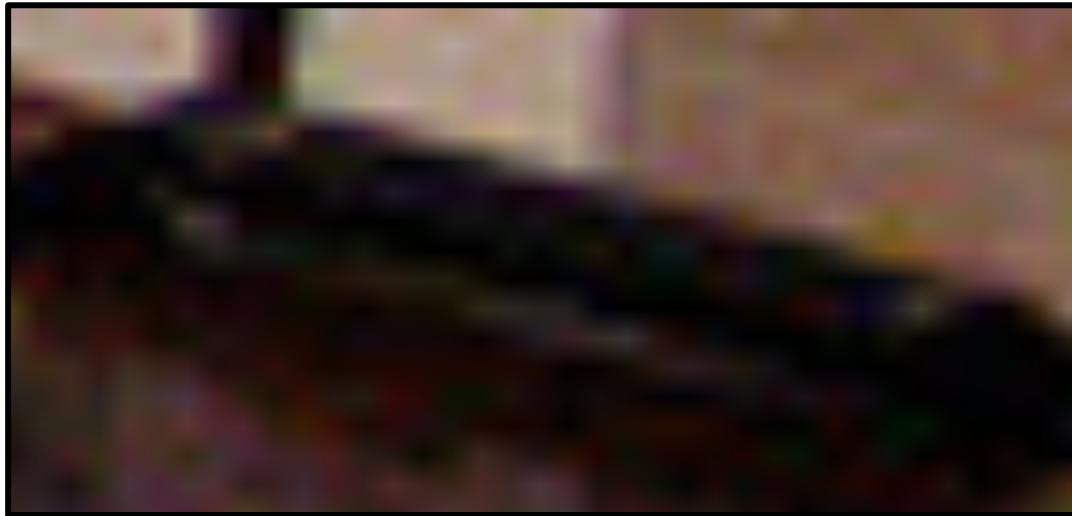
blur



# Why Is This Task Hard?

*Small patch*

## What's this? (No Cheating!)



(1) Keyboard?

(2) Hammer?

(3) Old cell phone?

(4) Xbox controller?

# Why Is This Task Hard?



Image credit: COCO dataset

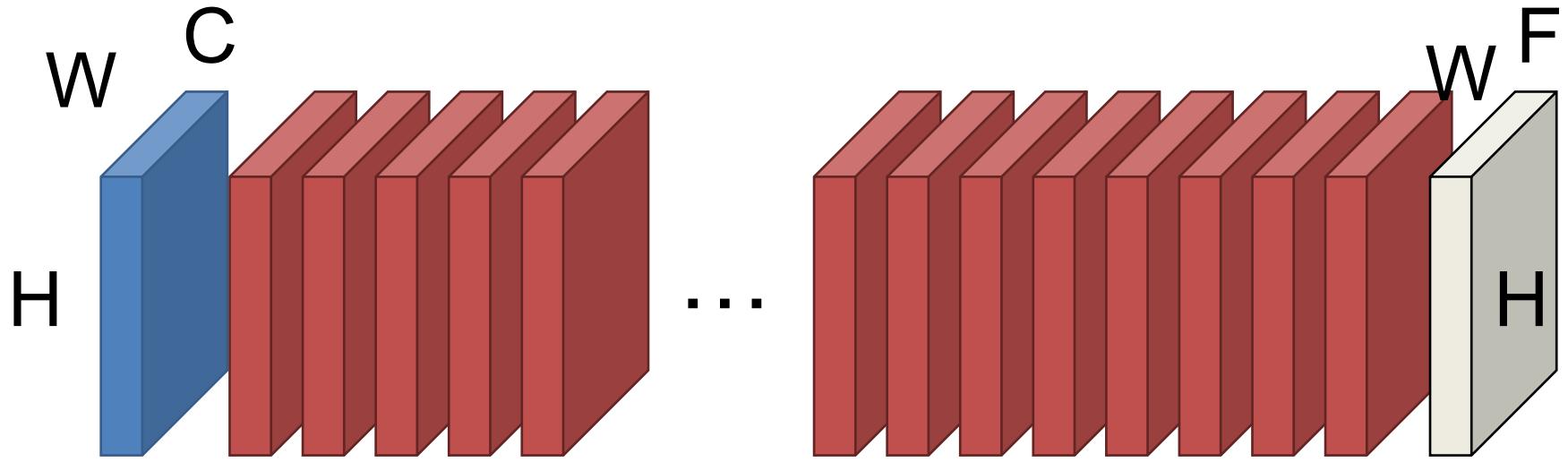
# Why Is This Task Hard?

- Low-resolution features
- Need large context window

# Why Is This Task Hard?

- It's helpful to see two “wrong” ways to do this.

# Why Not Stack Convolutions?

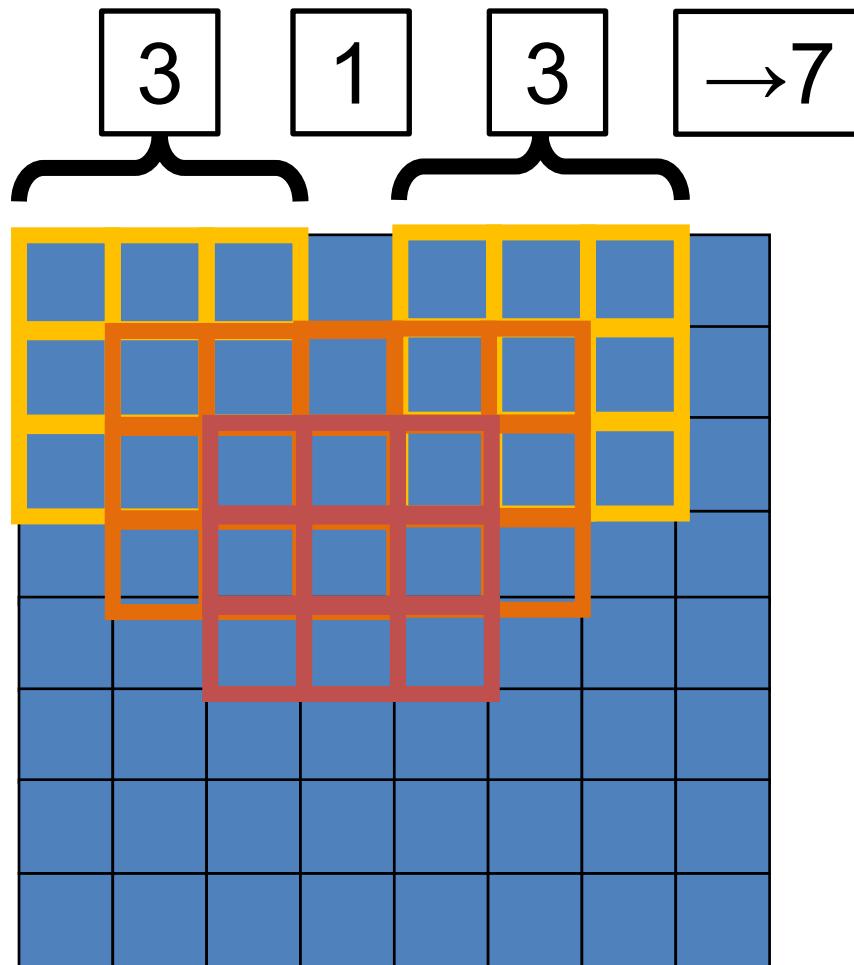


$n$  3x3 convs have a receptive field of  $2n+1$  pixels

**How many convolutions until  $\geq 200$  pixels?**

100

# 3x3 Filters

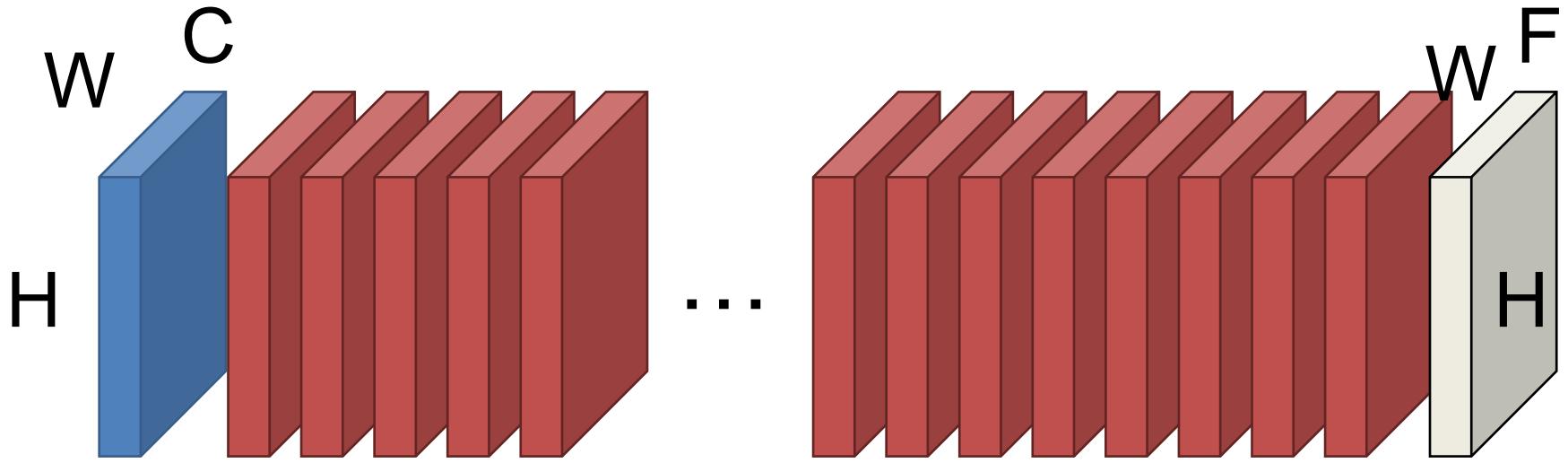


3x3 filter followed by  
3x3 filter followed by  
3x3 filter



Filter with  $7 \times 7$   
receptive field  
 $2n+1$

# Why Not Stack Convolutions?



Suppose 200 3x3 filters/layer,  $H=W=400$

Storage/layer/image:  $200 * 400 * 400 * 4 \text{ bytes} = 122\text{MB}$

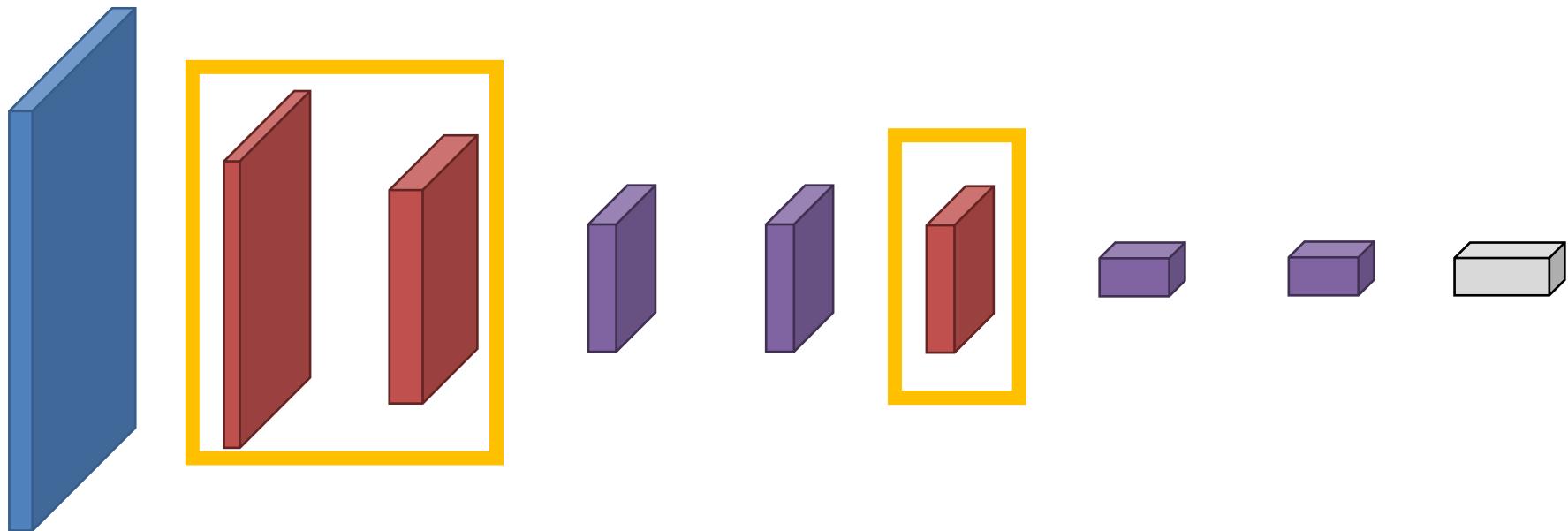
**Uh oh!\***

\*100 layers, batch size of 20 = 238GB of memory!

Any Solutions?

# Hence Need Low-Resolution

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227 3	55x55 96	27x27 256	13x13 384	13x13 384	13x13 256	1x1 4096	1x1 4096	1x1 1000



Need to reduce resolution to save memory.  
1/8<sup>th</sup> resolution → 1/64 memory

# Hence Need Low-Resolution

Input	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	FC 6	FC 7	Output
227x227 3	55x55 96	27x27 256	13x13 384	13x13 384	13x13 256	1x1 4096	1x1 4096	1x1 1000



Problem: Segmentation output is also low-resolution

# If Memory's the Issue...

Crop out every sub-window and predict the label in the middle.

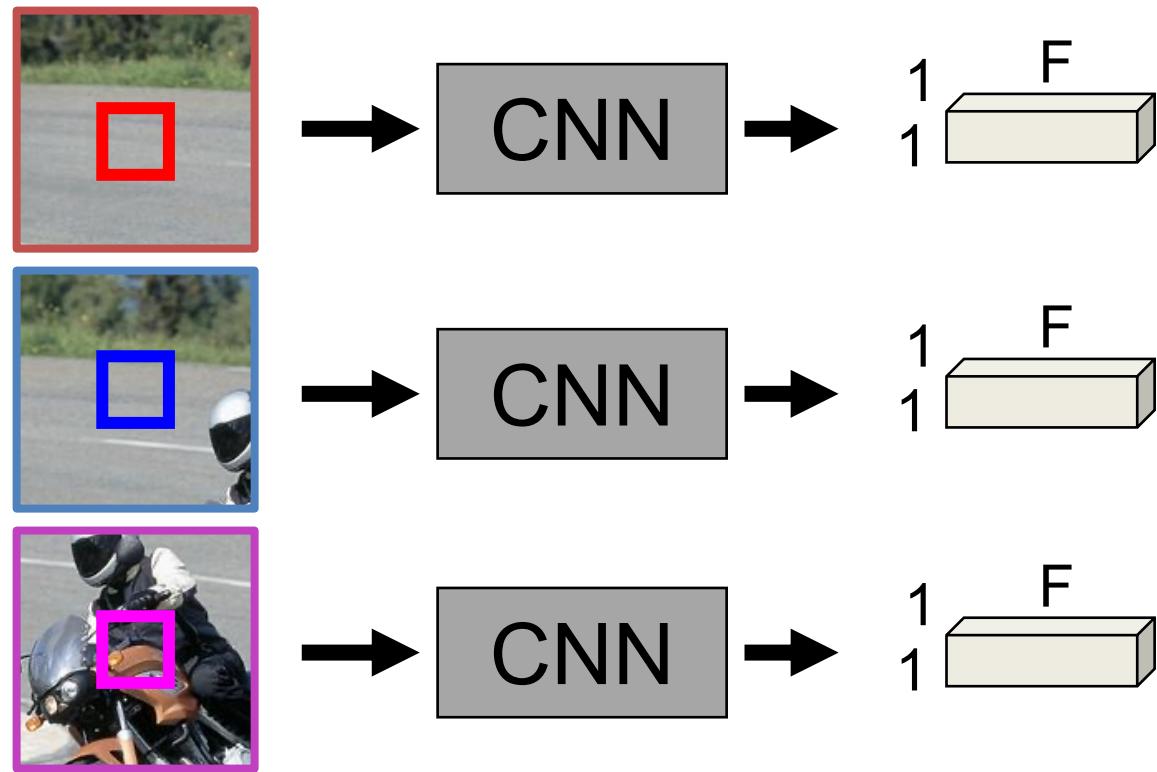
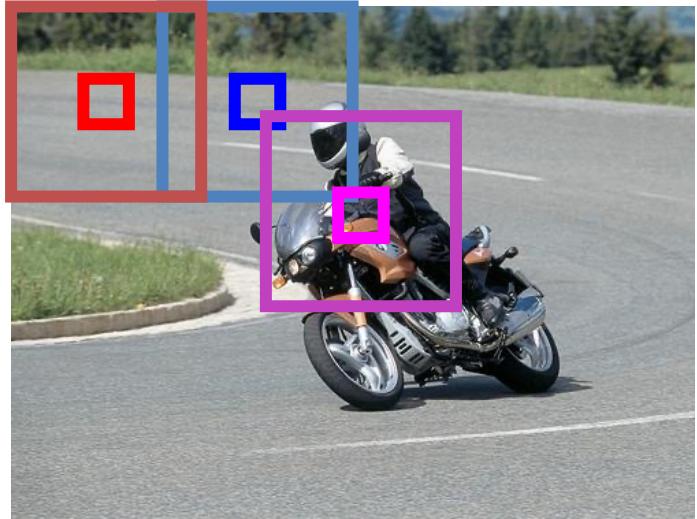


Image credit: PASCAL VOC, Everingham et al.

# If Memory's the Issue...

Crop out every sub-window and predict the label in the middle.



What are these patches? Paved or not?

Context is very important! Need a large receptive fields

# The Big Issue

We need to:

1. Have large receptive fields to figure out what we're looking at
2. Not waste a ton of time or memory while doing so

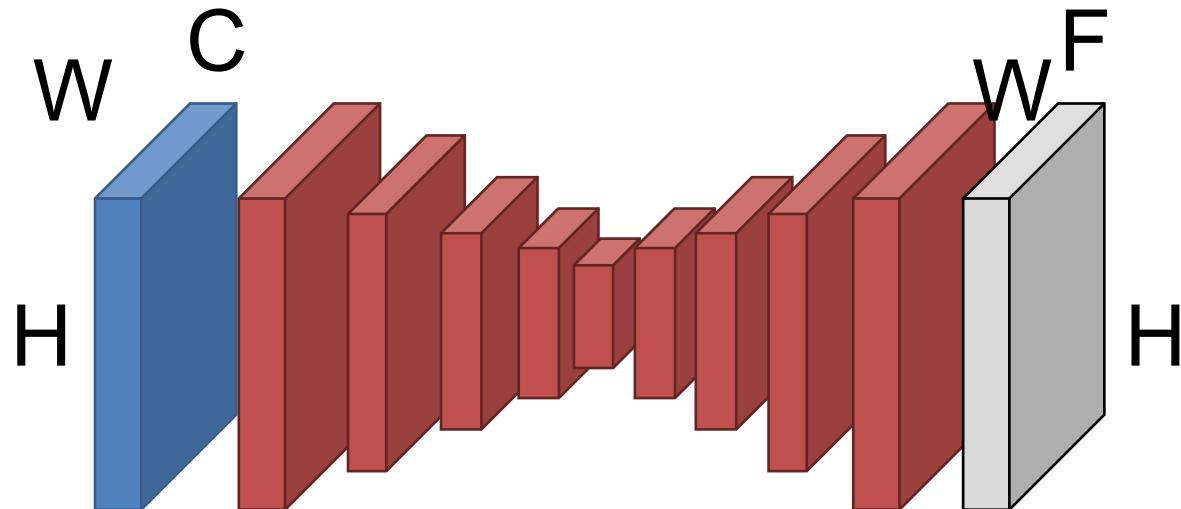
These two objectives are in total conflict

# Encoder-Decoder

Key idea: First **downsample** towards middle of network. Then **upsample** from middle.

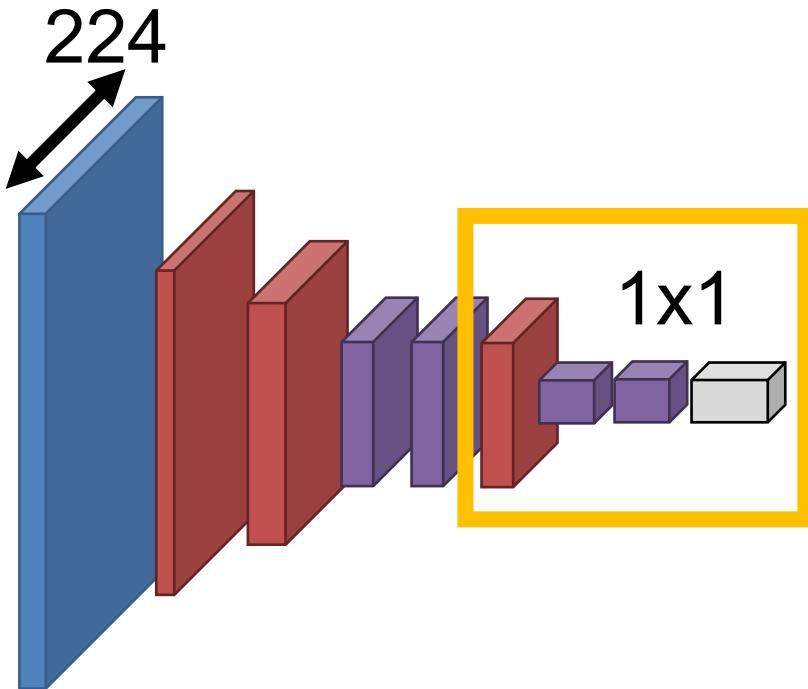
**How do we downsample?**

Strided-convolutions, pooling



# Fully Convolutional Network

Convnet that maps images to vectors



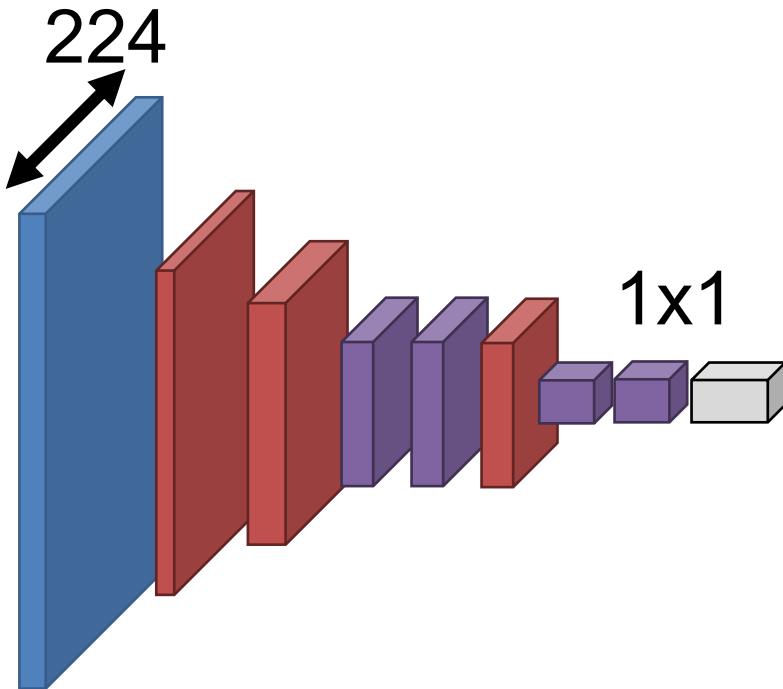
Fully-connected network has a fixed input-output dimensions

$$\text{purple cube} * \text{green cube} \rightarrow \text{purple cube}$$

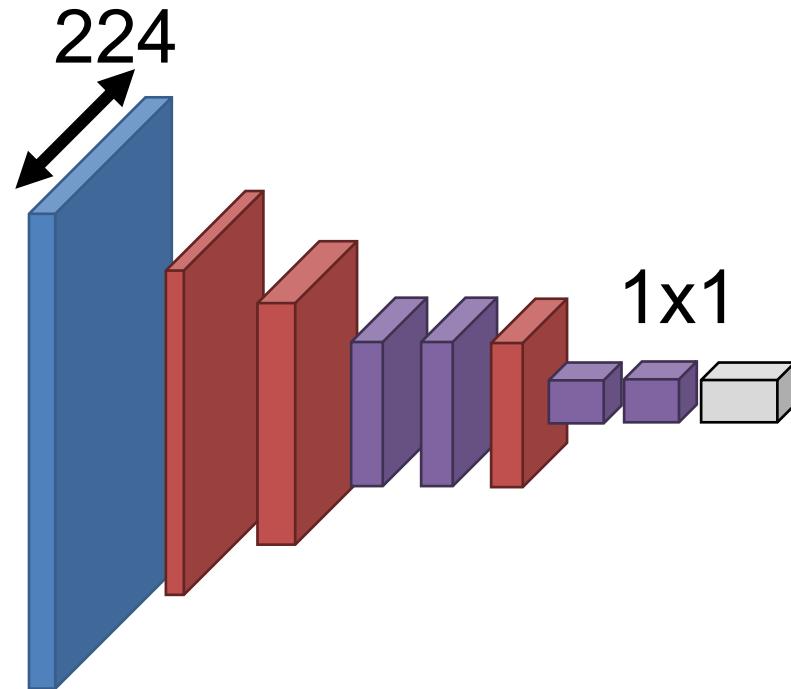
Recall that we can rewrite any vector-vector operations via 1x1 convolutions

# Fully Convolutional Network

Convnet that maps  
images to vectors



Convnet that maps  
images to images

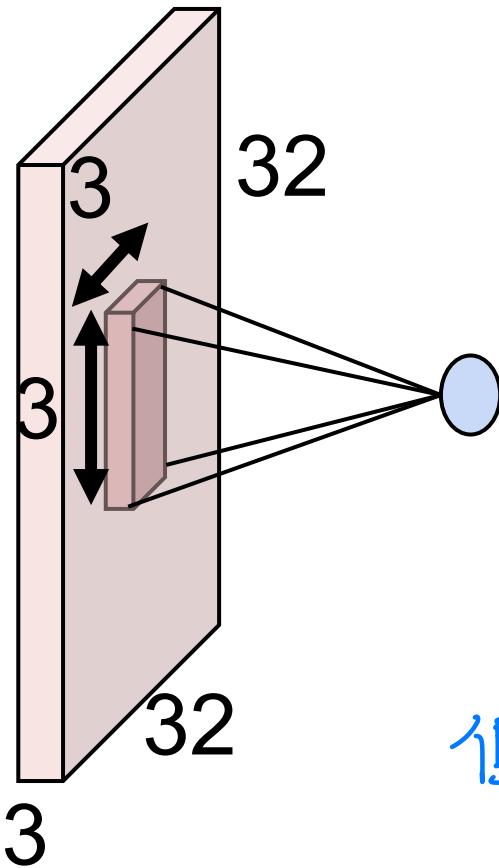


**What if we make the input bigger?**

可接受任何大小的 input.

# Convolution Layer

## How big is the output?

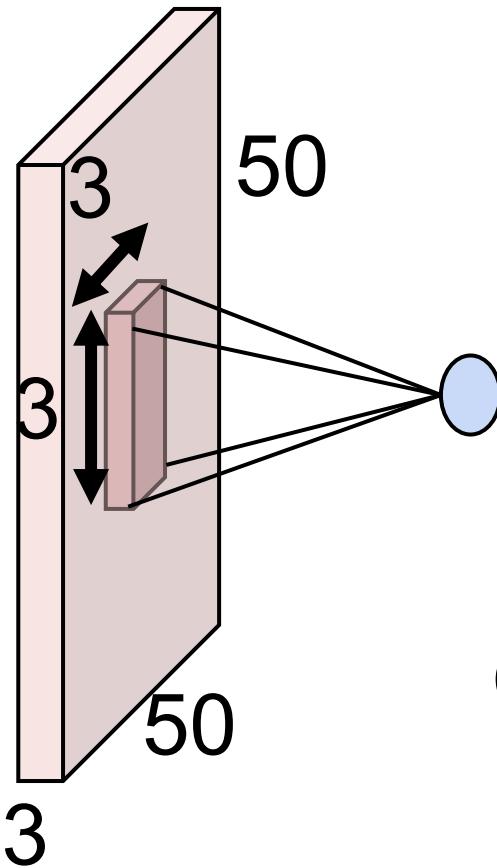


Height?  $32 - 3 + 1 = 30$   
Width?  $32 - 3 + 1 = 30$

但子C 的 input size是定好的.

# Convolution Layer

**How big is the output?**



Height?  $50 - 3 + 1 = 48$   
Width?  $50 - 3 + 1 = 48$

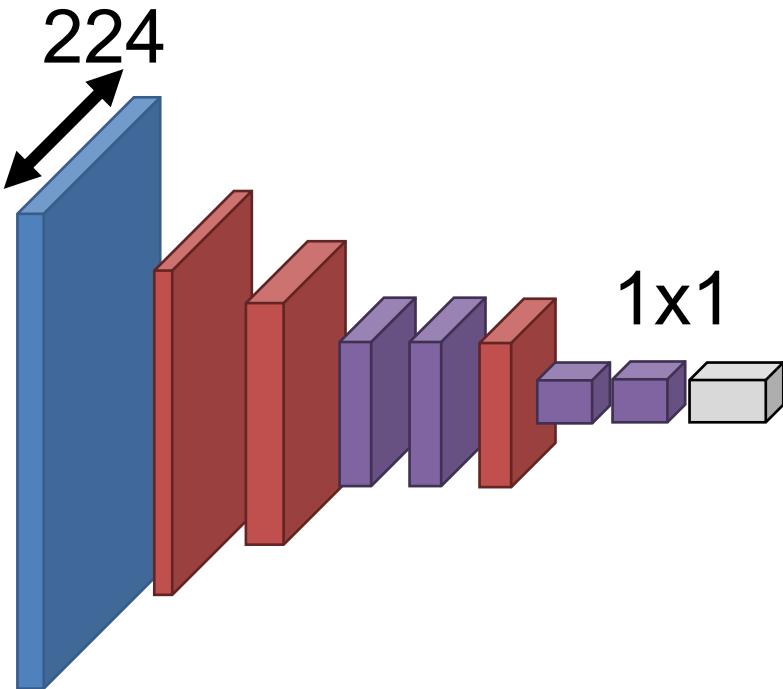
Can apply convolution layers to  
an image of any resolution

if your CNN architecture is fully convolutional Network

# Fully Convolutional Network

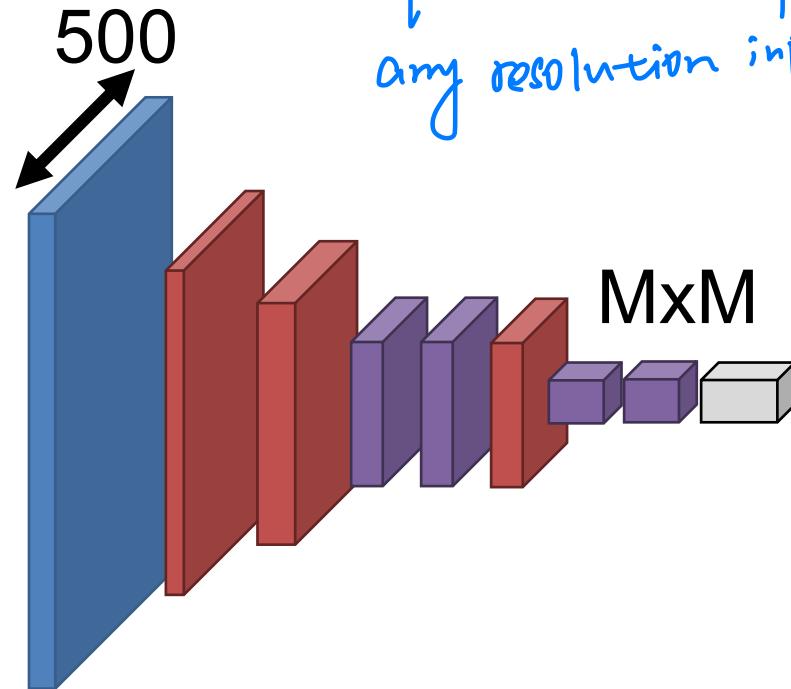
then even  
you train it with

Convnet that maps  
images to vectors



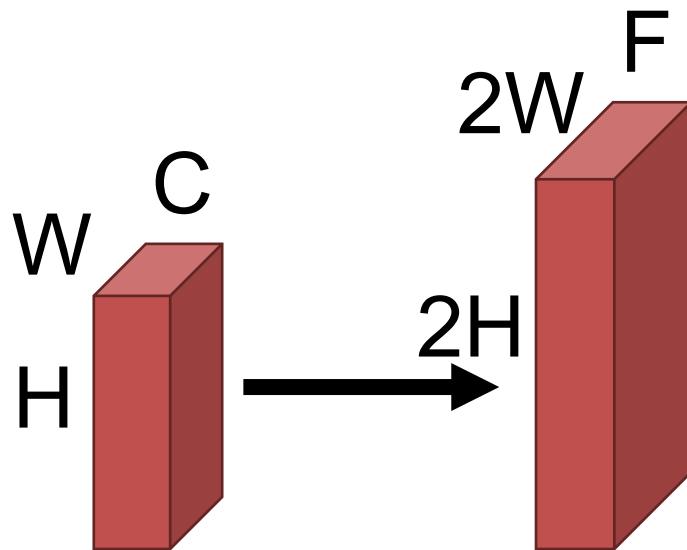
Convnet that maps  $224 \times 224$   
images to images

You can use it for  
any resolution input image



Since it's convolution, can reuse an image network  
E.g., train on 224 ImageNet, test on COCO Dataset

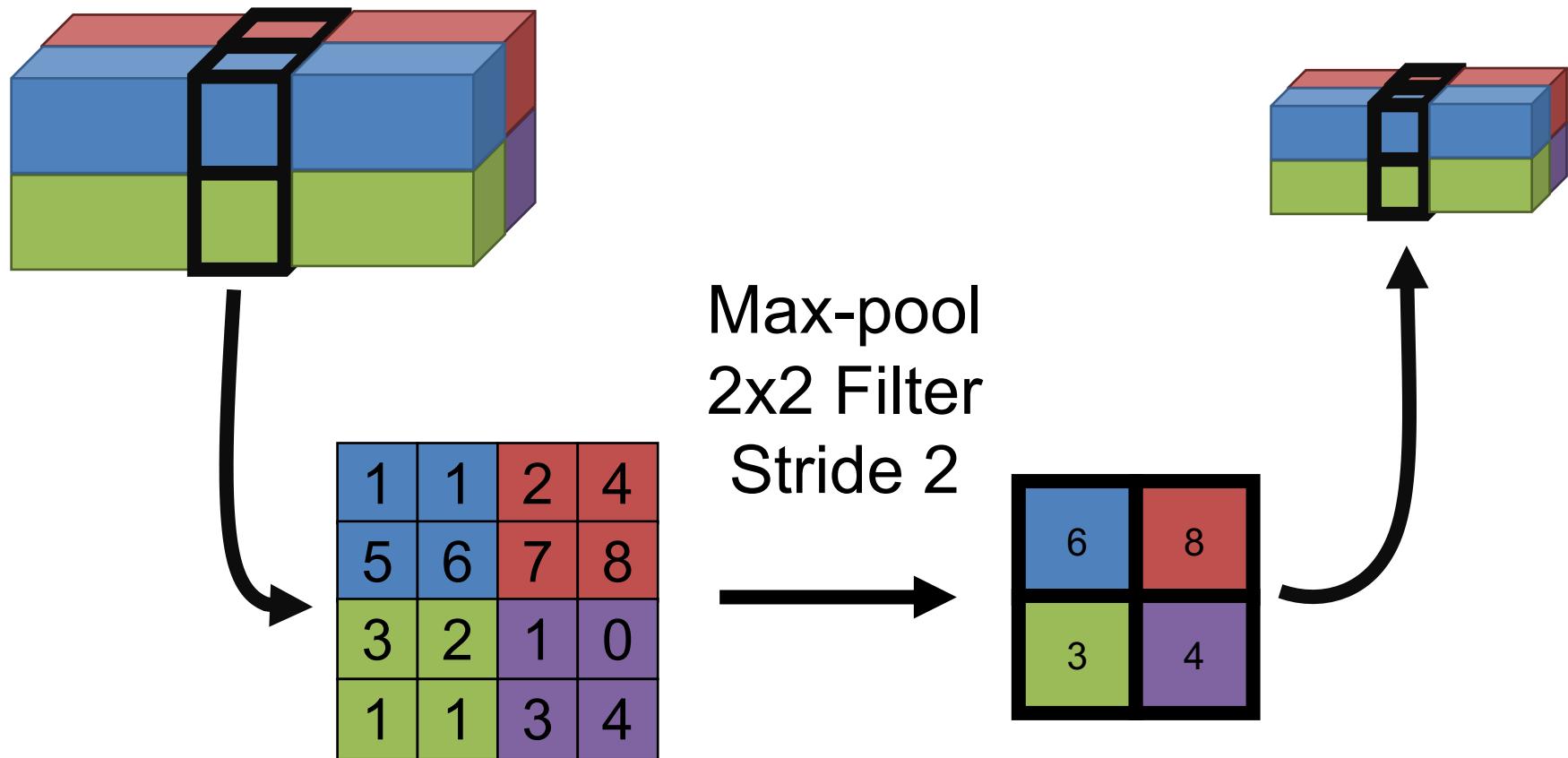
# How Do We **Upsample**?



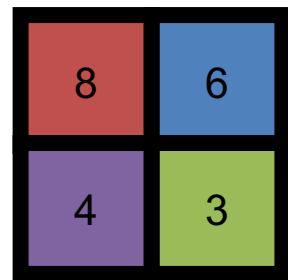
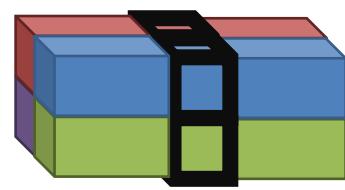
Do the opposite of how we downsample:

1. Pooling → “Unpooling”
2. Convolution → “Transpose Convolution”

# Recall: Pooling



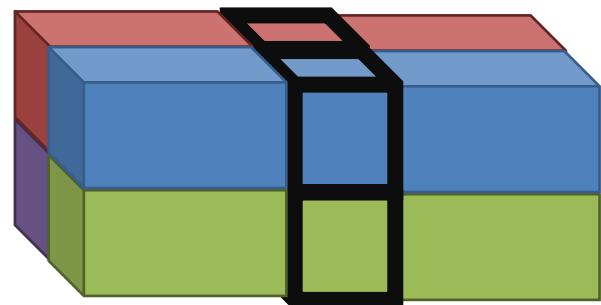
# Now: Unpooling



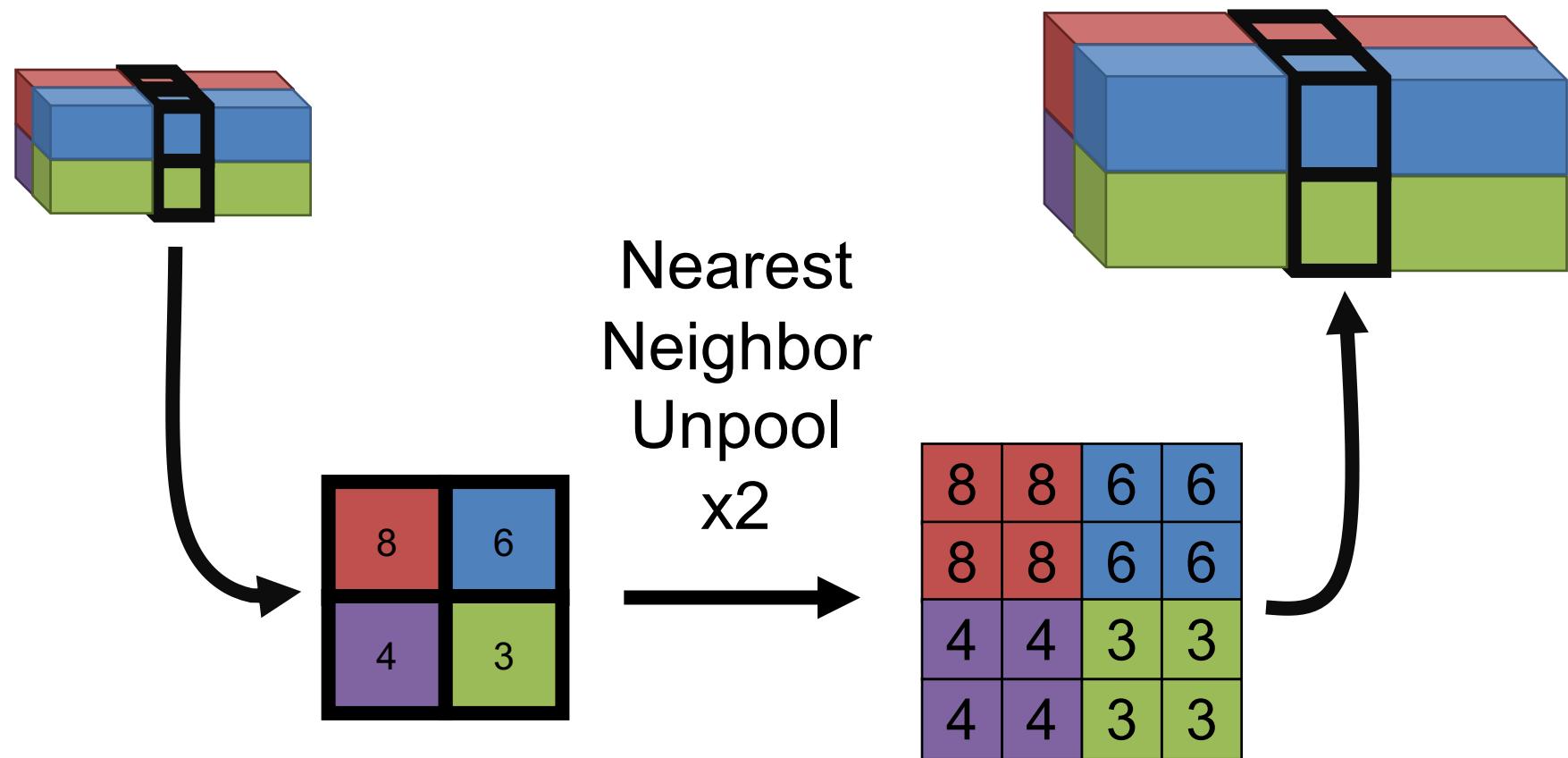
Nearest  
Neighbor  
Unpool  
 $\times 2$



8	8	6	6
8	8	6	6
4	4	3	3
4	4	3	3



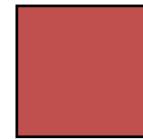
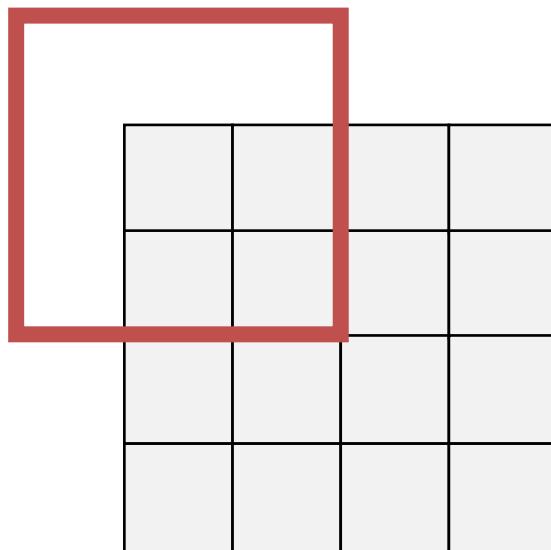
# Now: Unpooling



Other interpolations possible: bilinear, bicubic, etc

# Recall: Convolution

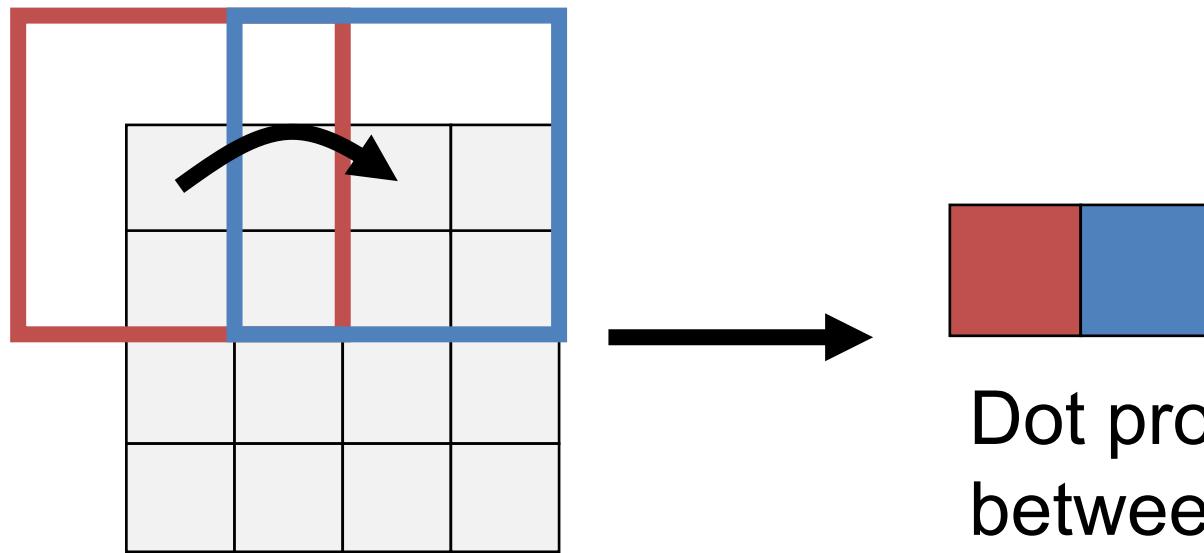
3x3 Convolution, Stride 2, Pad 1



Dot product  
between filter f  
and input

# Recall: Convolution

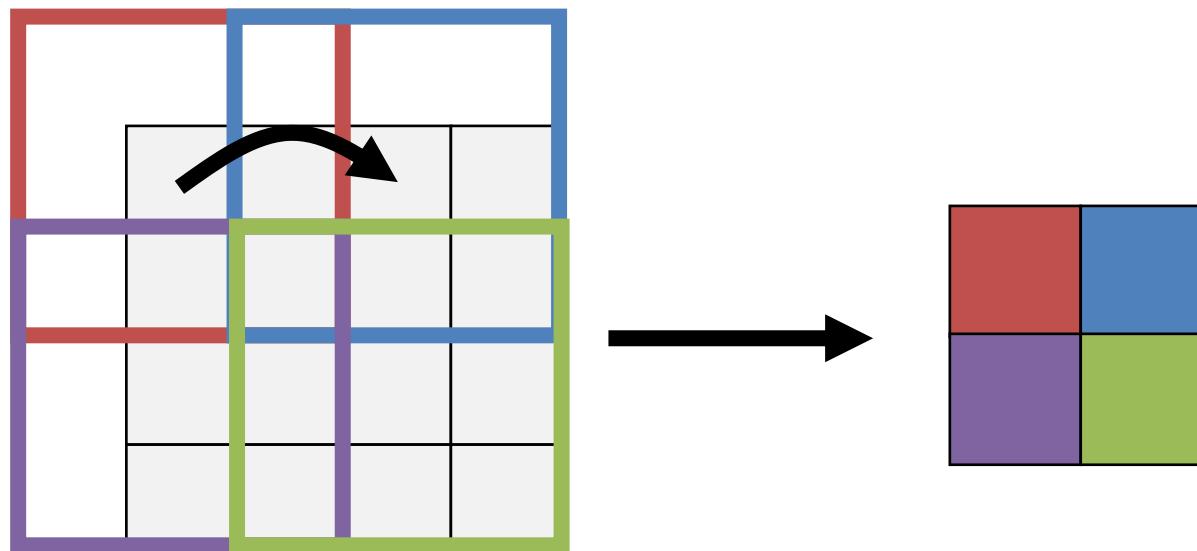
3x3 Convolution, Stride 2, Pad 1



Dot product  
between filter f  
and input

# Recall: Convolution

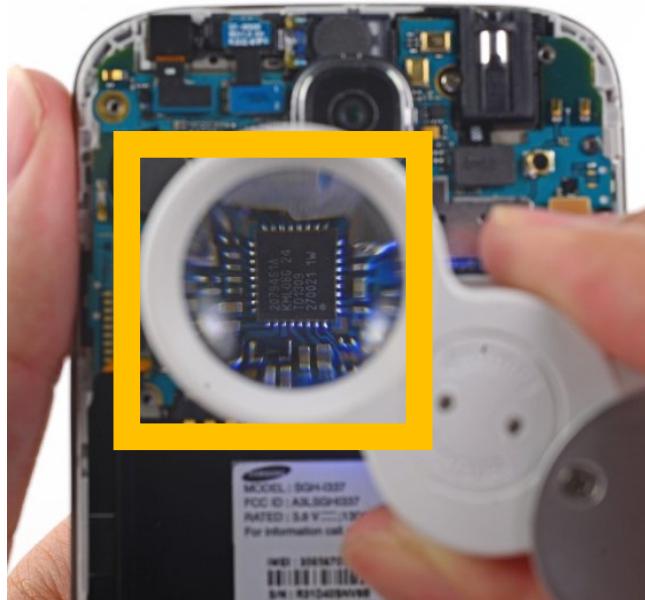
3x3 Convolution, Stride 2, Pad 1



# Transpose Convolution

## Convolution

Filter: little (sliding) lens  
that looks at a pixel.



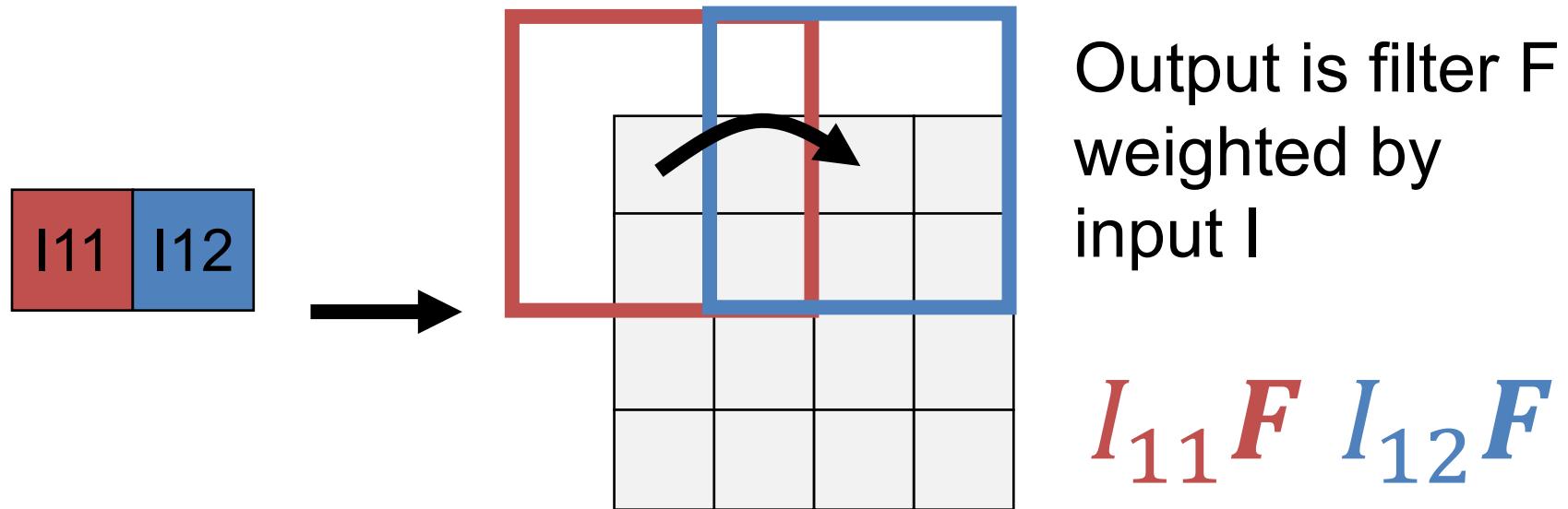
## Transpose Conv.

Filter: tiles used to  
make image



# Transpose Convolution

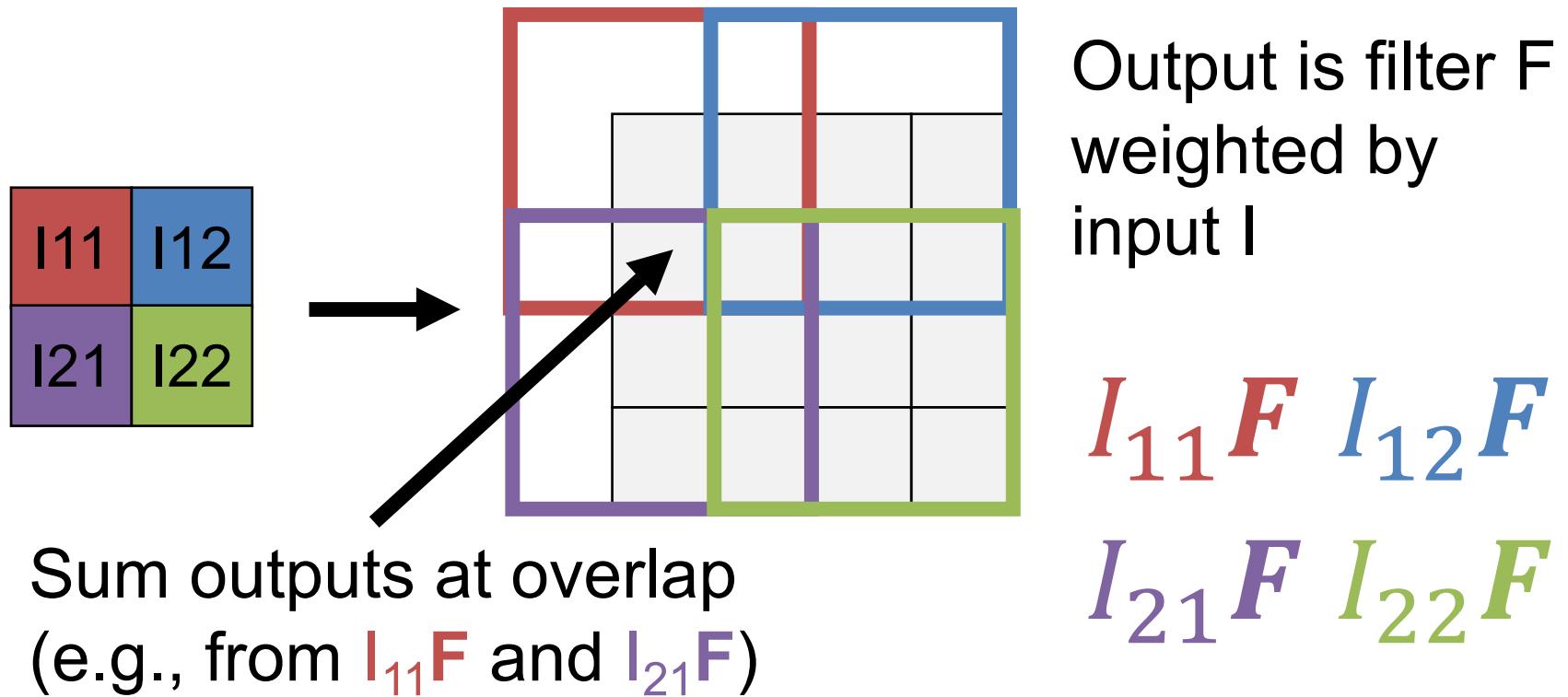
3x3 Transpose Convolution, Stride 2, Pad 1



Each low-res pixel maps to a tile

# Transpose Convolution

3x3 Transpose Convolution, Stride 2, Pad 1



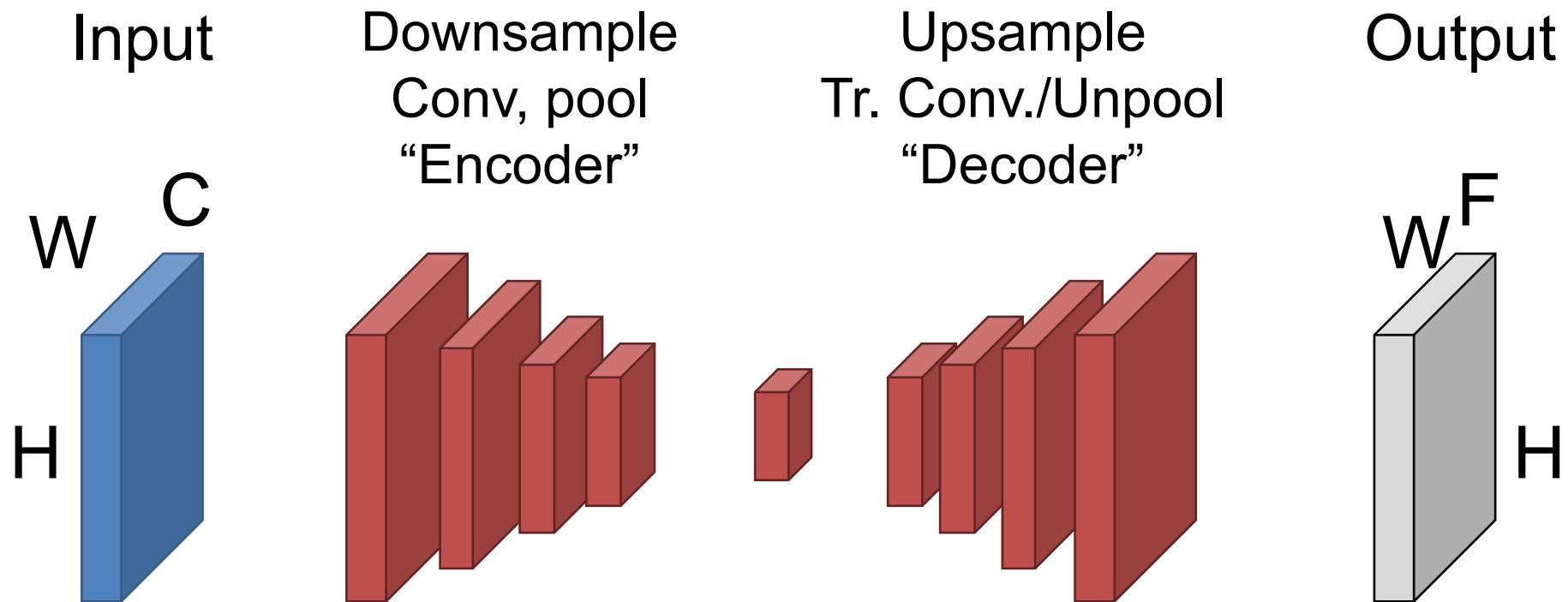
# Transpose Convolution

Demo

Resource (for the curious)

# Putting it Together

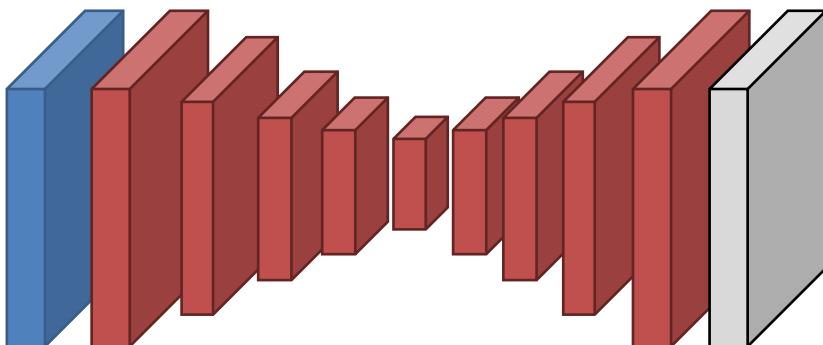
Convolutions + pooling downsample/compress/encode  
Transpose convs./unpoolings upsample/uncompress/decode



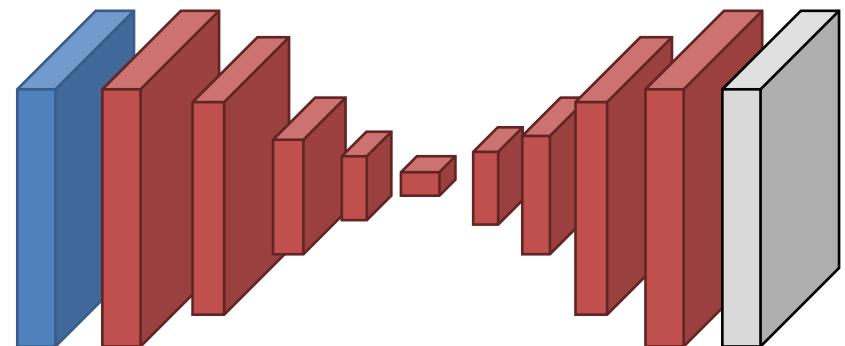
# Putting It Together – Block Sizes

- Networks come in lots of forms
- **Don't take any block sizes literally.**
- Often (not always) keep some spatial resolution

Encode to spatially smaller tensor, then decode.

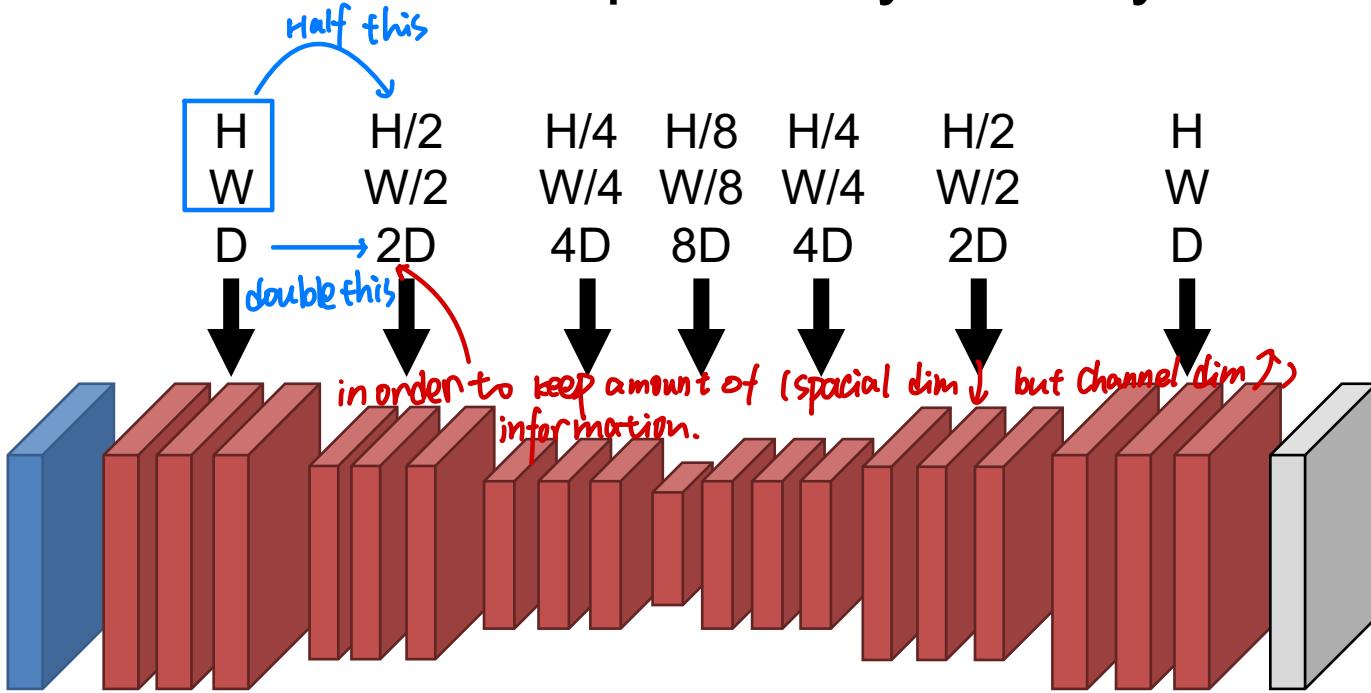


Encode to 1D vector then decode



# Putting It Together – Block Sizes

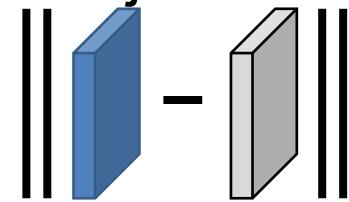
- Often multiple layers at each spatial resolution.
  - Often halve spatial resolution and double feature depth every few layers



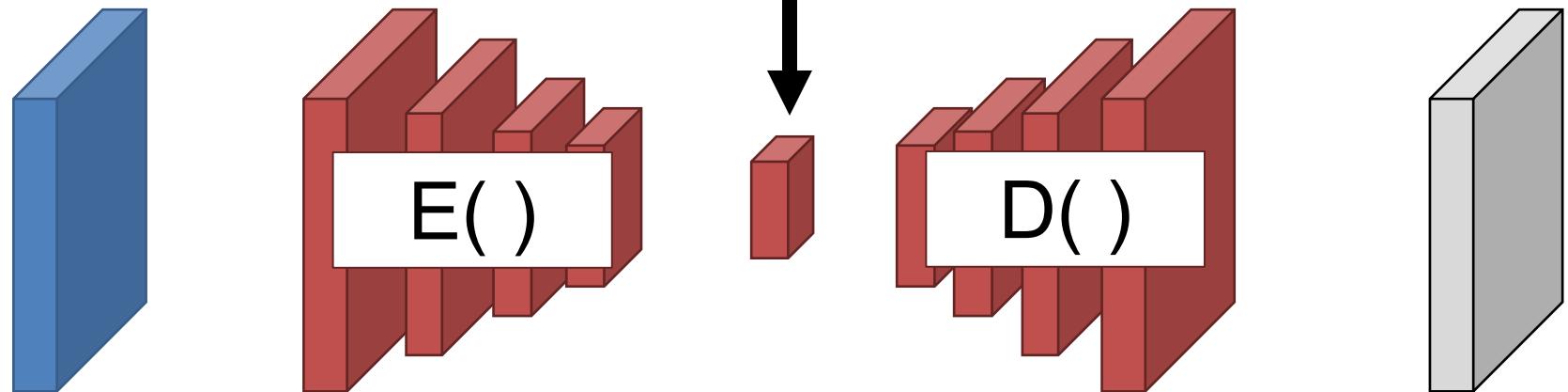
# An Aside: Autoencoders

Network compresses input to “bottleneck”, decodes it back to input.  
Abstract latent space contrast to “Data Space.”

Objective:

$$\|D(E(X)) - X\|$$


Bottleneck/  
Latent Space/  
Latent Code



Loss function is to make sure that all information is preserved from your input to output.

# Walking the Latent Space\*

Linear Interpolation in the latent space



\*In the interest of honesty in advertising: not an autoencoder, but a similar method with the same goal of learning a latent space

Result from Park et al. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. CVPR 2019

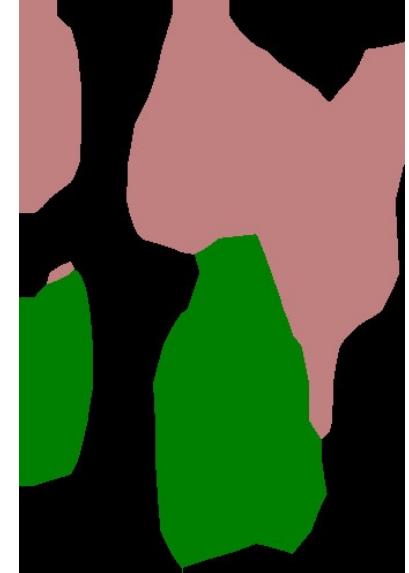
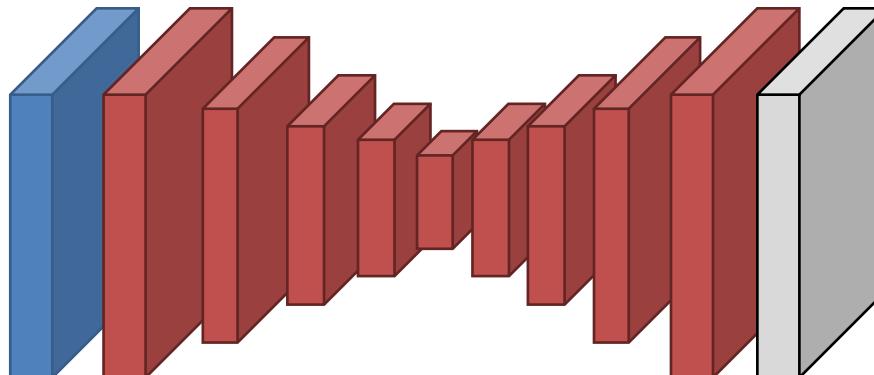
# Missing Spatial Details

While the output *is*  $H \times W$ , just upsampling often produces results without details/not aligned with the image.

**Why?**

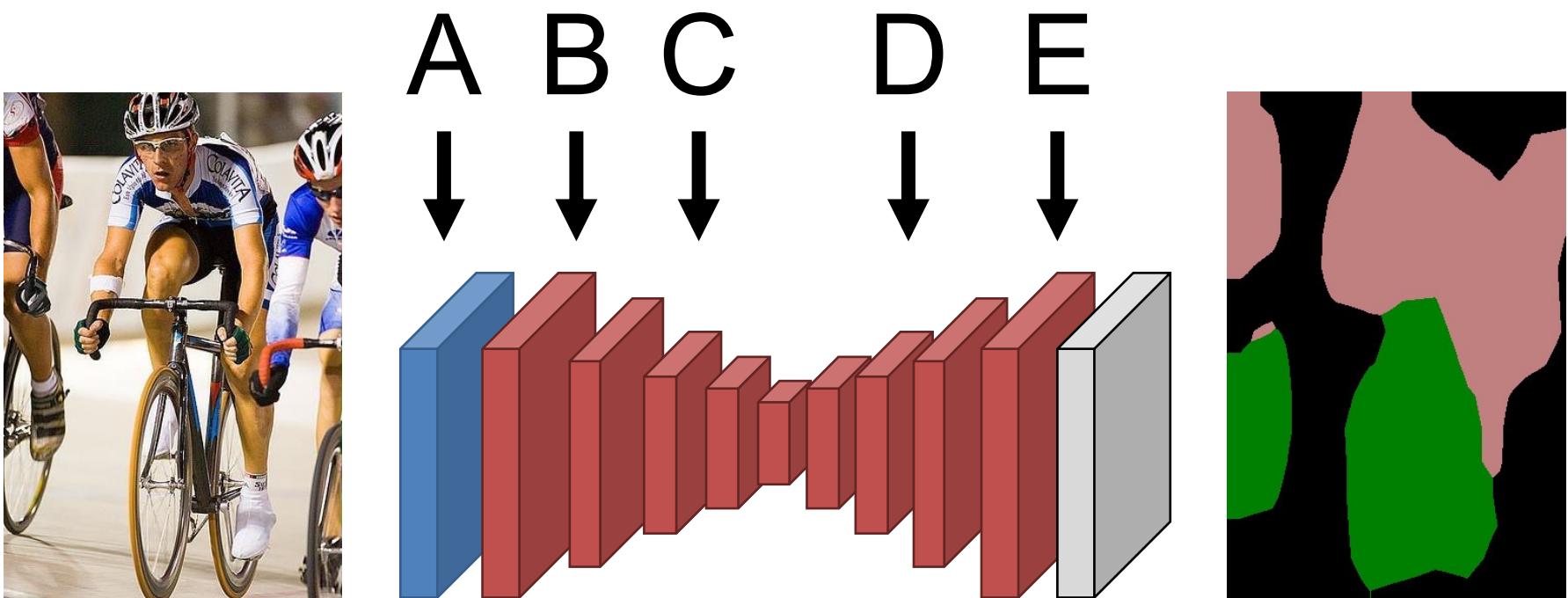


Information about details  
lost when downsampling!



# Missing Spatial Details

Where is the most useful information about the high-frequency details of the image?



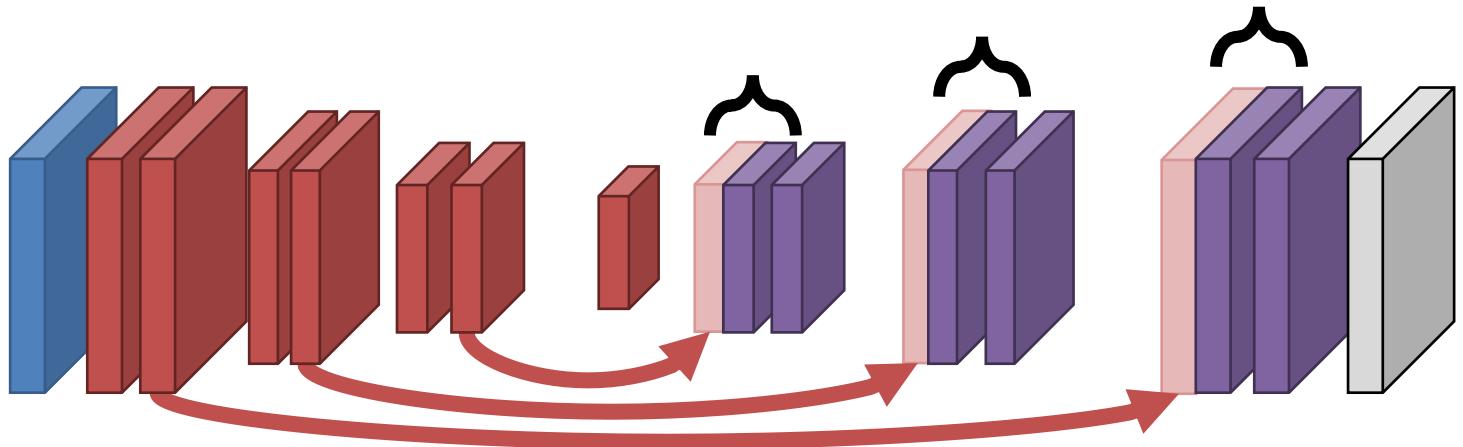
# Missing Spatial Details

How do you send details forward in the network?

You copy the activations forward.

Subsequent layers at the same resolution figure out how to fuse things.

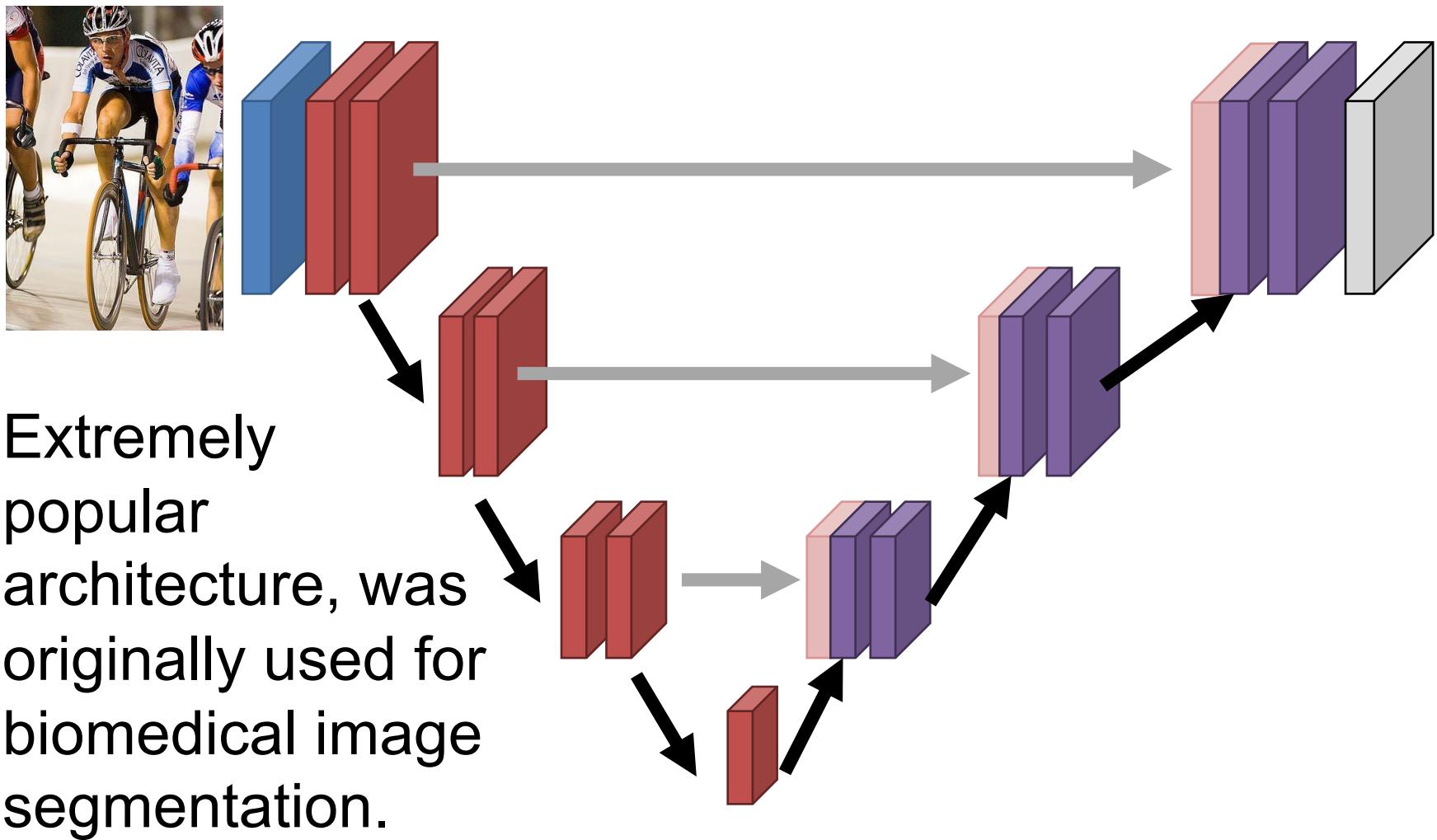
Often called Residual Connections as in ResNet



Copy

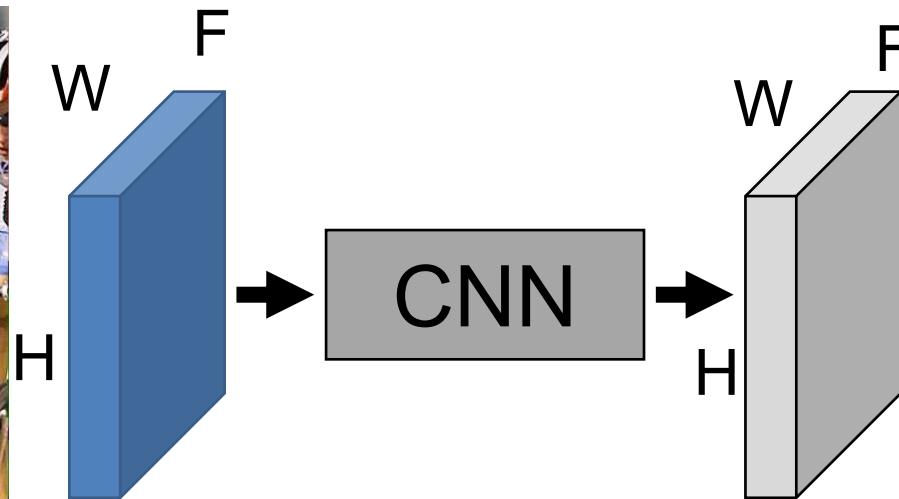
Result from Long et al. *Fully Convolutional Networks For Semantic Segmentation*. CVPR 2014

# U-Net

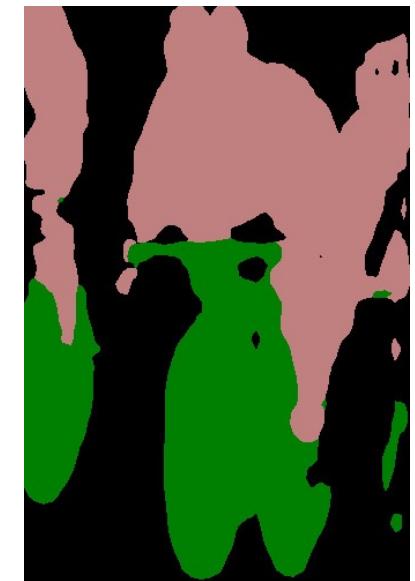


# Evaluating Pixel Labels

Input  
Image



Predicted  
Classes



**How do we convert final  $H \times W \times F$  into labels?**

argmax over labels

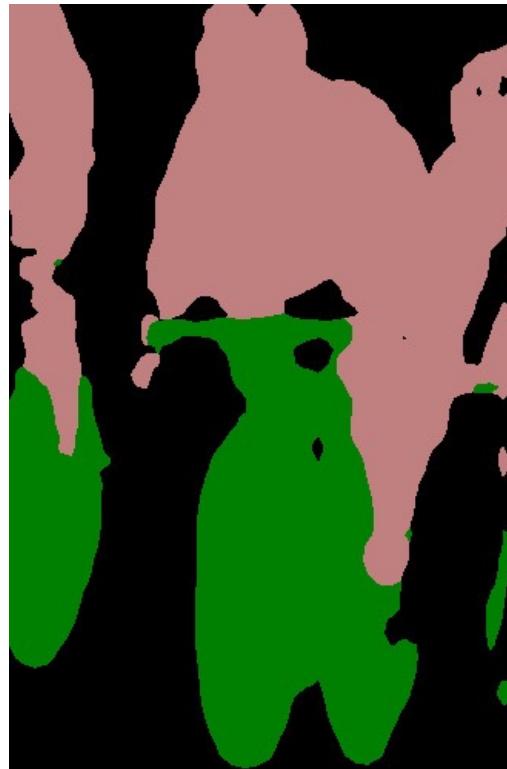
# Evaluating Semantic Segmentation

Given predictions, how well did we do?

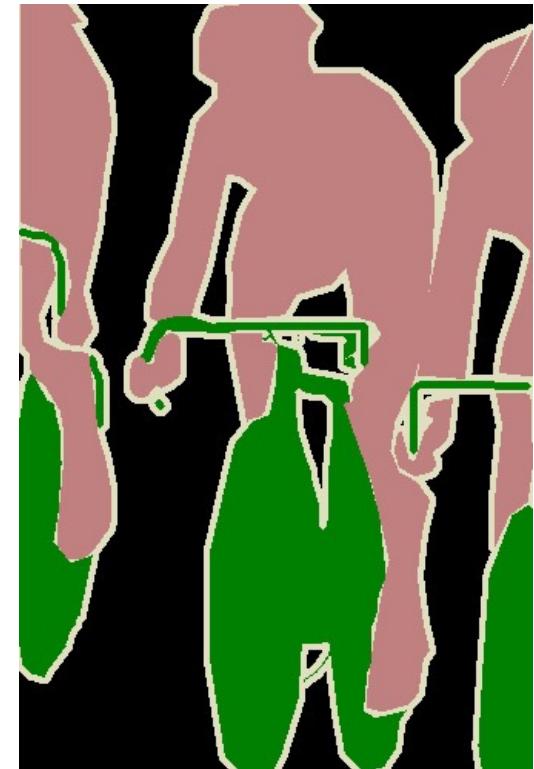
Input



Prediction ( $\hat{y}$ )



Ground-Truth ( $y$ )



# Evaluating Semantic Segmentation

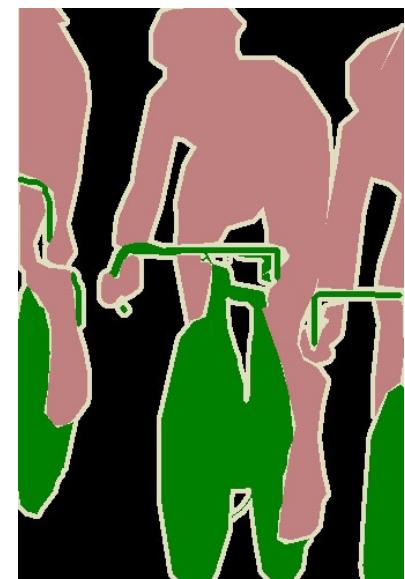
Prediction and ground-truth are images where each pixel is one of F classes.

Accuracy:  $\text{mean}(\hat{y} = y)$

Prediction  
( $\hat{y}$ )



Ground-Truth  
( $y$ )



# Evaluating Semantic Segmentation

Prediction and ground-truth are images where each pixel is one of F classes.

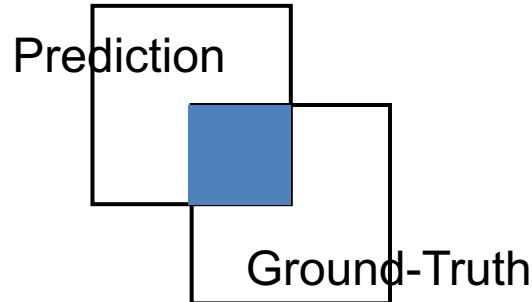
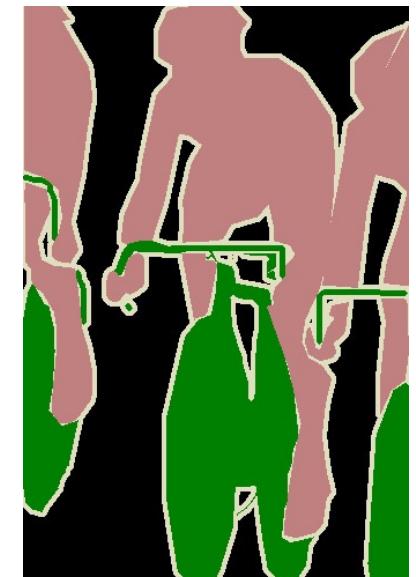
Accuracy:  $\text{mean}(\hat{y} = y)$

Intersection over union,  
averaged over classes

Prediction  
 $(\hat{y})$



Ground-Truth  
 $(y)$



# Evaluating Semantic Segmentation

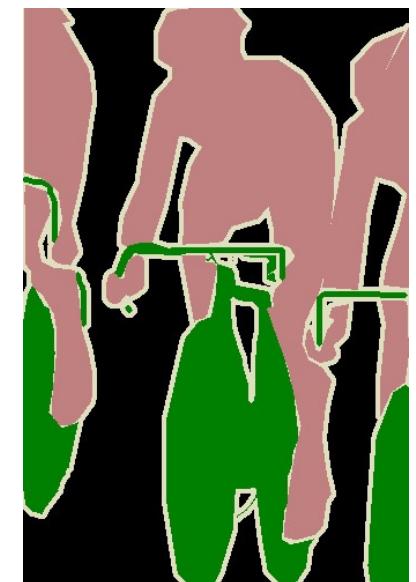
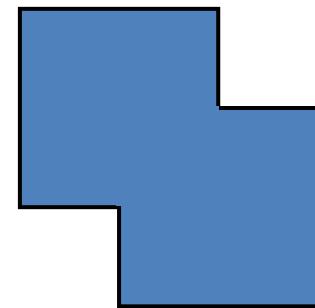
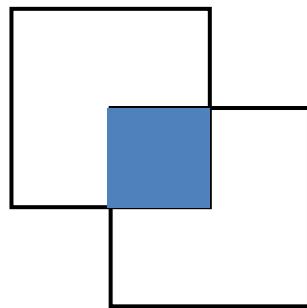
Prediction and ground-truth are images where each pixel is one of F classes.

Accuracy:  $\text{mean}(\hat{y} = y)$

Prediction  
( $\hat{y}$ )

Ground-Truth  
( $y$ )

Intersection over union,  
averaged over classes



# Next Time

- Detecting Objects (drawing boxes around them)

# More Info

# Why “Transpose Convolution”?

Can write convolution as matrix-multiply  
Input: 4, Filter: 3, Stride: 1, Pad: 1

$$\begin{matrix} a & b & c & d \end{matrix} * \begin{matrix} x & y & z \end{matrix} = \begin{matrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{matrix}$$

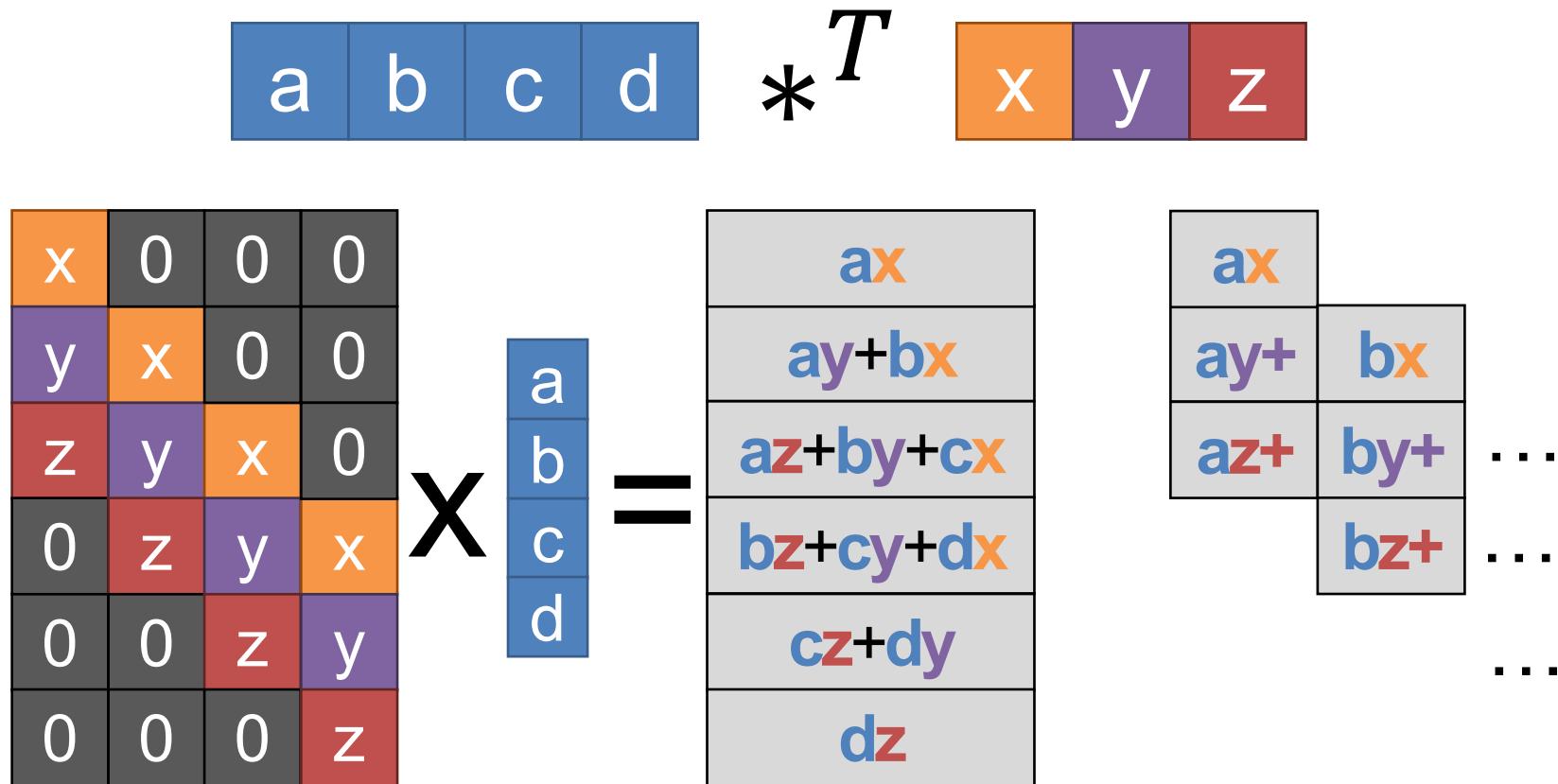
The diagram illustrates the computation of a 4x1 input vector  $\begin{matrix} a & b & c & d \end{matrix}$  with a 3x1 filter  $\begin{matrix} x & y & z \end{matrix}$ . The result is a 4x1 output vector where each element is the sum of products between corresponding elements of the input and filter. The input is zero-padded to 6 elements. The filter is also zero-padded to 6 elements. The resulting output is:

$ay + bz$
$ax + by + cz$
$bx + cy + dz$
$cx + dy$

# Why “Transpose Convolution”?

Transpose convolution is convolution transposed

$$\begin{matrix} a & b & c & d \end{matrix} *^T \begin{matrix} x & y & z \end{matrix}$$



x	0	0	0
y	x	0	0
z	y	x	0
0	z	y	x
0	0	z	y
0	0	0	z

$\times$  =

ax
ay+bx
az+by+cx
bz+cy+dx
cz+dy
dz

$\begin{matrix} ax \\ ay+bx \\ az+by+cx \\ bz+cy+dx \\ cz+dy \\ dz \end{matrix}$

$\begin{matrix} ax \\ ay+bx \\ az+by+cx \\ bz+cy+dx \\ cz+dy \\ dz \end{matrix}$

$\dots$

$\dots$

$\dots$